

(12) **United States Patent**
Fiske

(10) **Patent No.:** **US 12,124,850 B2**
(45) **Date of Patent:** **Oct. 22, 2024**

(54) **HIDING STABLE MACHINE INSTRUCTIONS
IN NOISE**

(71) Applicant: **Michael Stephen Fiske**, San Francisco,
CA (US)

(72) Inventor: **Michael Stephen Fiske**, San Francisco,
CA (US)

(73) Assignee: **Fiske Software LLC**, San Francisco,
CA (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 372 days.

(21) Appl. No.: **17/516,722**

(22) Filed: **Nov. 2, 2021**

(65) **Prior Publication Data**
US 2022/0058022 A1 Feb. 24, 2022

Related U.S. Application Data

(63) Continuation-in-part of application No. 16/442,455,
filed on Jun. 15, 2019, now Pat. No. 11,171,934, and
a continuation-in-part of application No. 14/953,300,
filed on Nov. 28, 2015, now Pat. No. 10,360,395.

(60) Provisional application No. 63/210,500, filed on Jun.
15, 2021, provisional application No. 62/092,795,
filed on Dec. 16, 2014, provisional application No.
62/085,338, filed on Nov. 28, 2014.

(51) **Int. Cl.**
G06F 9/30 (2018.01)
G06N 10/00 (2022.01)

(52) **U.S. Cl.**
CPC **G06F 9/30058** (2013.01); **G06F 9/30054**
(2013.01); **G06N 10/00** (2019.01)

(58) **Field of Classification Search**
CPC G06F 9/30058; G06F 9/30054; G06N
10/00

See application file for complete search history.

(56) **References Cited**

U.S. PATENT DOCUMENTS

11,509,461 B2 * 11/2022 Schat G06F 9/30178
2002/0172179 A1 * 11/2002 Grimsrud G06F 9/30156
370/441

(Continued)

OTHER PUBLICATIONS

Fiske, A Cryptographically Stable Computing Machine, 2022, Asso-
ciation for Information Systems, Conference: 55th Hawaii Interna-
tional Conference on System Sciences 2022, pp. 7566-7575 (Year:
2022).*

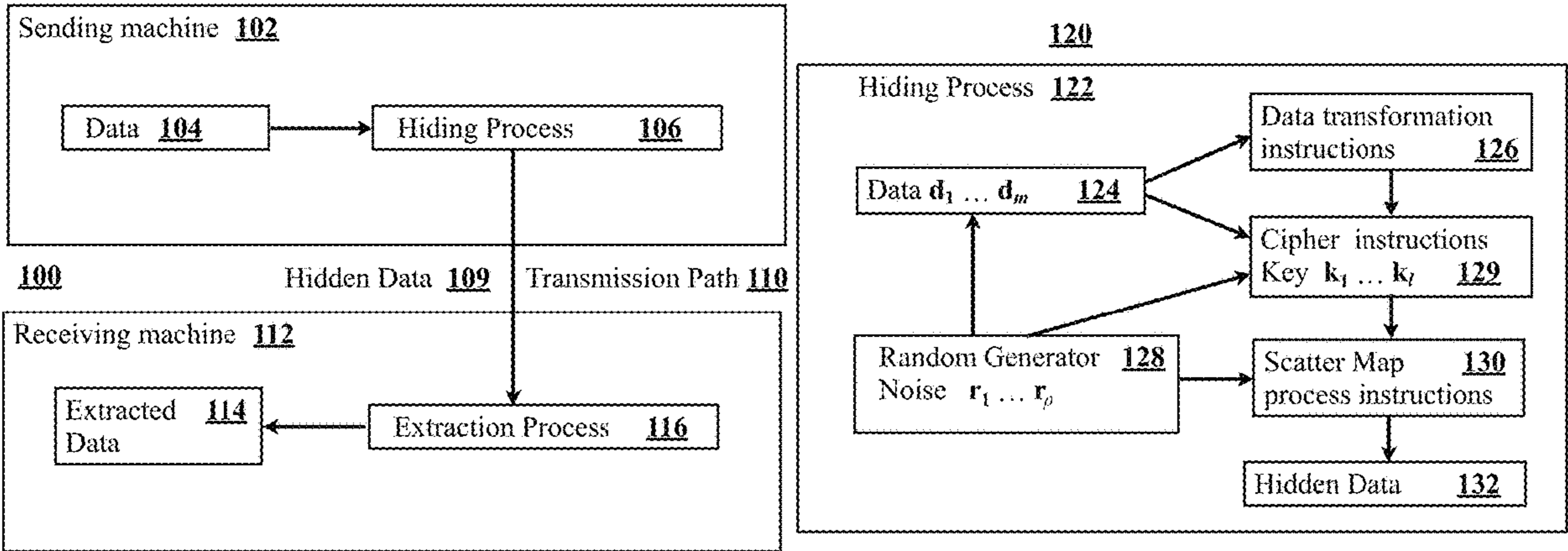
Primary Examiner — Stephen T Gundry

(57) **ABSTRACT**

Our machine architecture and machine procedures use
robustness, unpredictably and variability to hinder malware
infection. In some embodiments, our machine instruction
opcodes are randomized. The computing behavior of our
machine is structurally stable (invariant) to small changes
made to its machine instructions. Our invention expands the
engineering method of stability to a cryptographically stable
machine that is resistant to malware sabotage by an adver-
sary.

Our procedures use quantum randomness to build unpre-
dictable stable instructions. Our machine procedures can
execute just before running a program so that the computing
task can be performed with a different representation of its
instructions during each run. A process of hiding a key or
data inside of random noise is described that protects the
privacy of the machine instruction opcodes and operands. In
some embodiments, quantum randomness generates random
noise, using photonic emission with a light emitting diode.

20 Claims, 28 Drawing Sheets



(56)

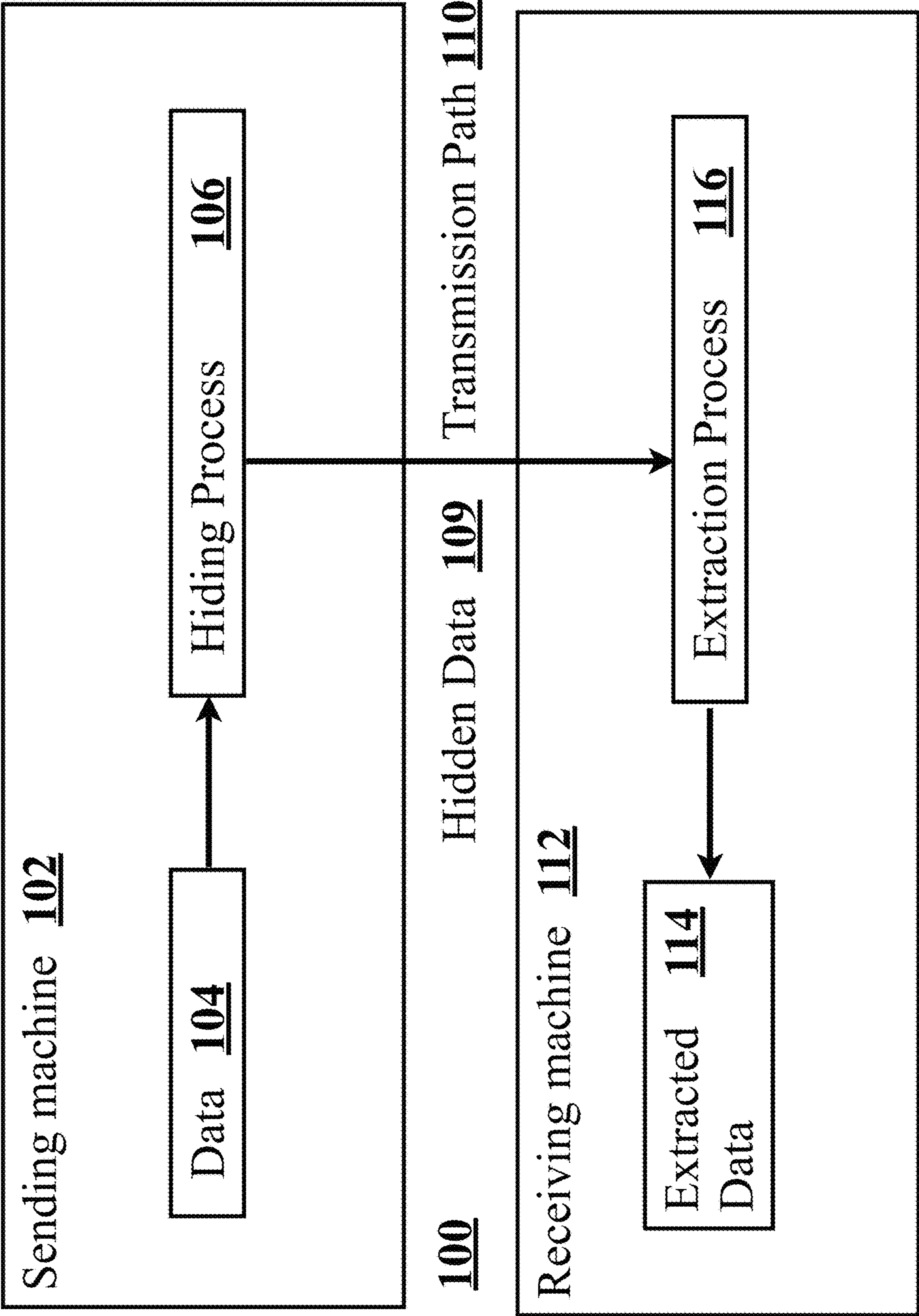
References Cited

U.S. PATENT DOCUMENTS

2010/0082719	A1 *	4/2010	Adir	G06F 7/582 708/250
2019/0312854	A1 *	10/2019	Fiske	H04L 9/0852
2020/0380344	A1 *	12/2020	Lie	G06N 3/08
2020/0380370	A1 *	12/2020	Lie	G06F 17/10
2021/0142155	A1 *	5/2021	James	G06N 3/045
2021/0182077	A1 *	6/2021	Chen	G06F 7/5443
2021/0234555	A1 *	7/2021	Alvarado Segovia	H03M 13/152
2021/0248453	A1 *	8/2021	Lauterbach	G06N 3/063
2021/0255860	A1 *	8/2021	Morrison	G06N 3/084
2021/0256362	A1 *	8/2021	Lie	G06N 3/047
2021/0351922	A1 *	11/2021	Schat	G06F 9/30178
2022/0058022	A1 *	2/2022	Fiske	H04L 9/0852
2022/0197658	A1 *	6/2022	Agron	G06F 9/30145
2022/0343136	A1 *	10/2022	Morrison	G06F 13/42
2023/0030988	A1 *	2/2023	Tee	H04L 41/22
2023/0069536	A1 *	3/2023	Morrison	G06N 3/063
2023/0071424	A1 *	3/2023	Kibardin	G06F 18/214

* cited by examiner

FIG. 1A



120 FIG. 1B

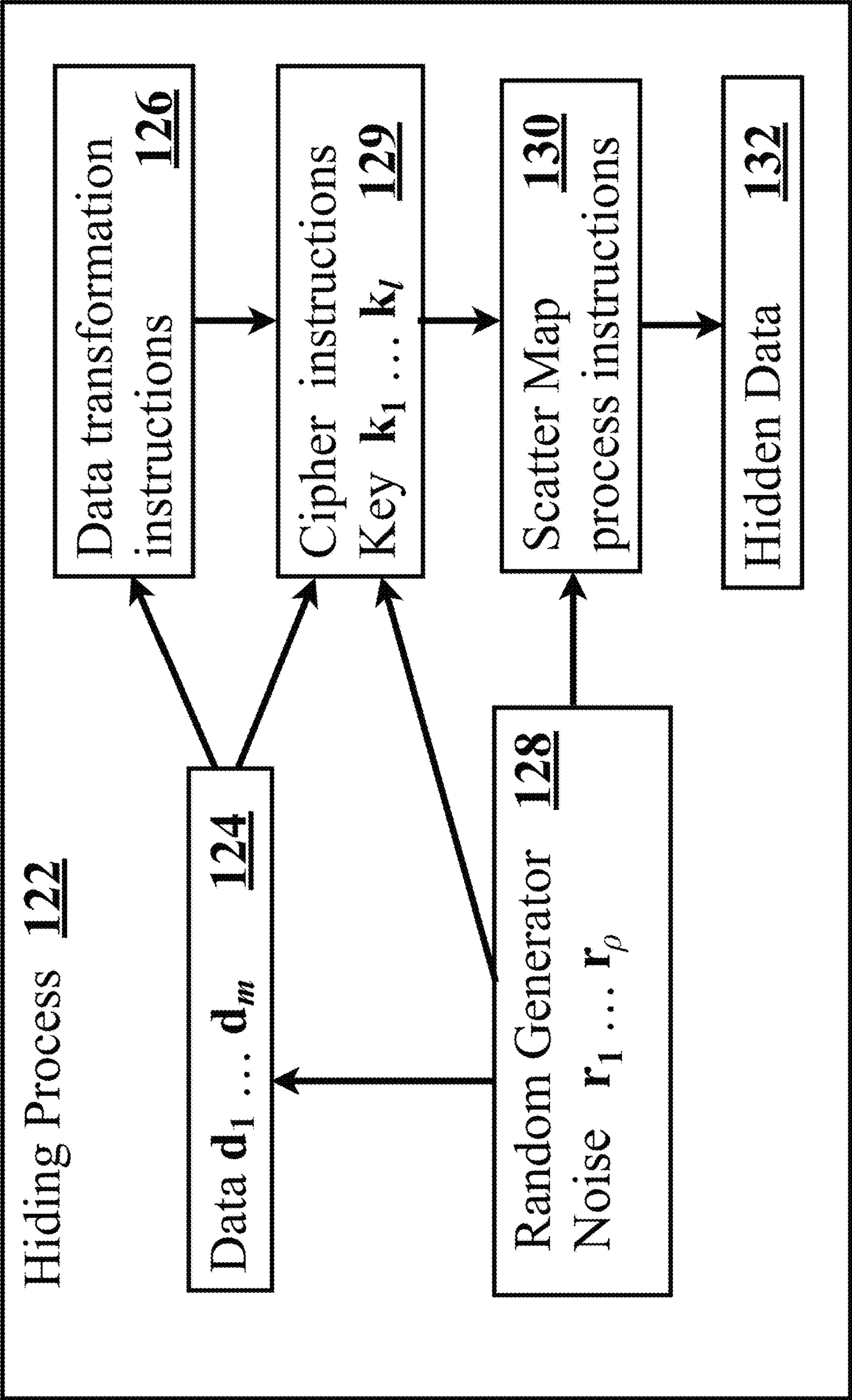
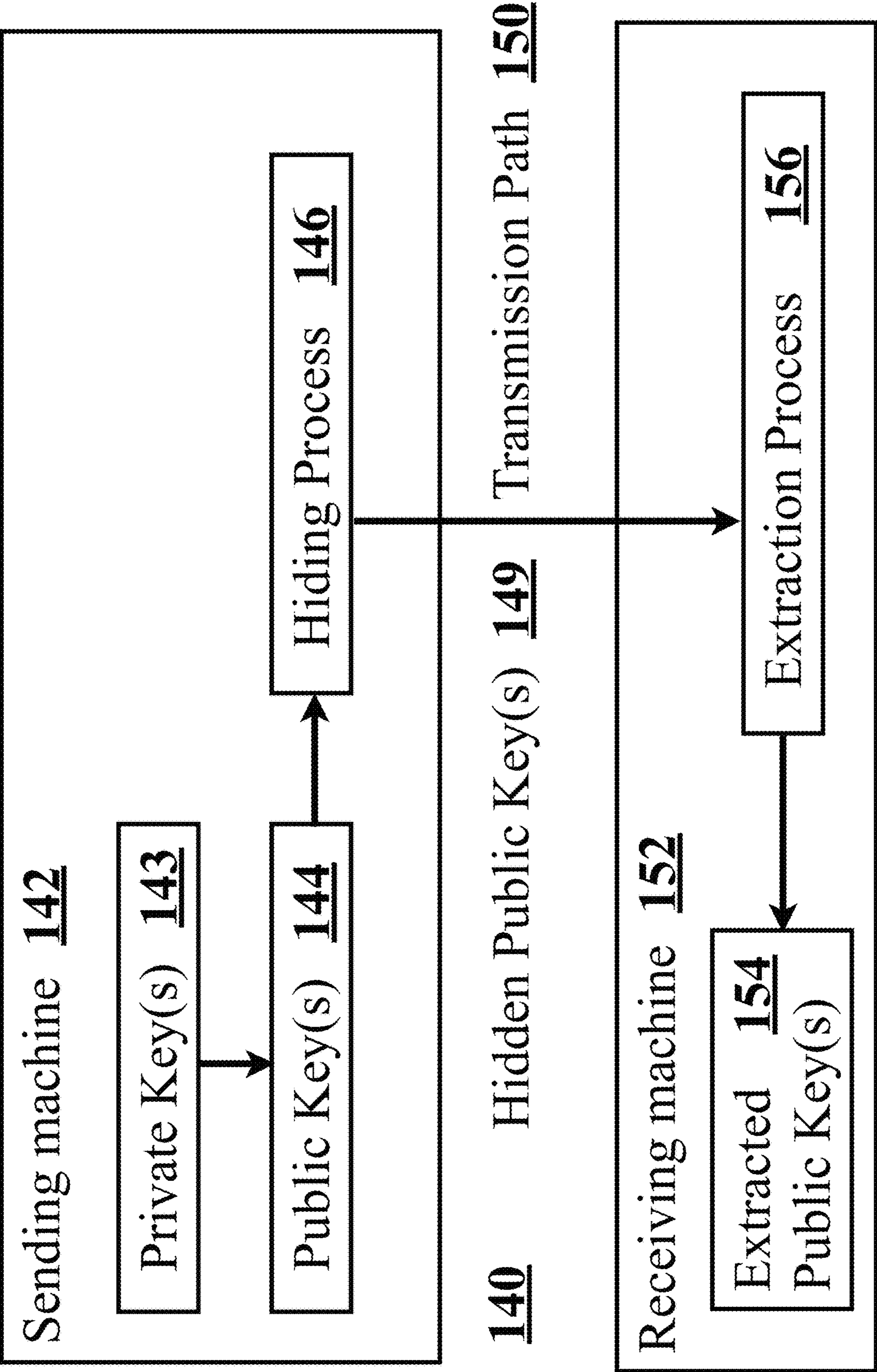


FIG. 1C



160 FIG. 1D

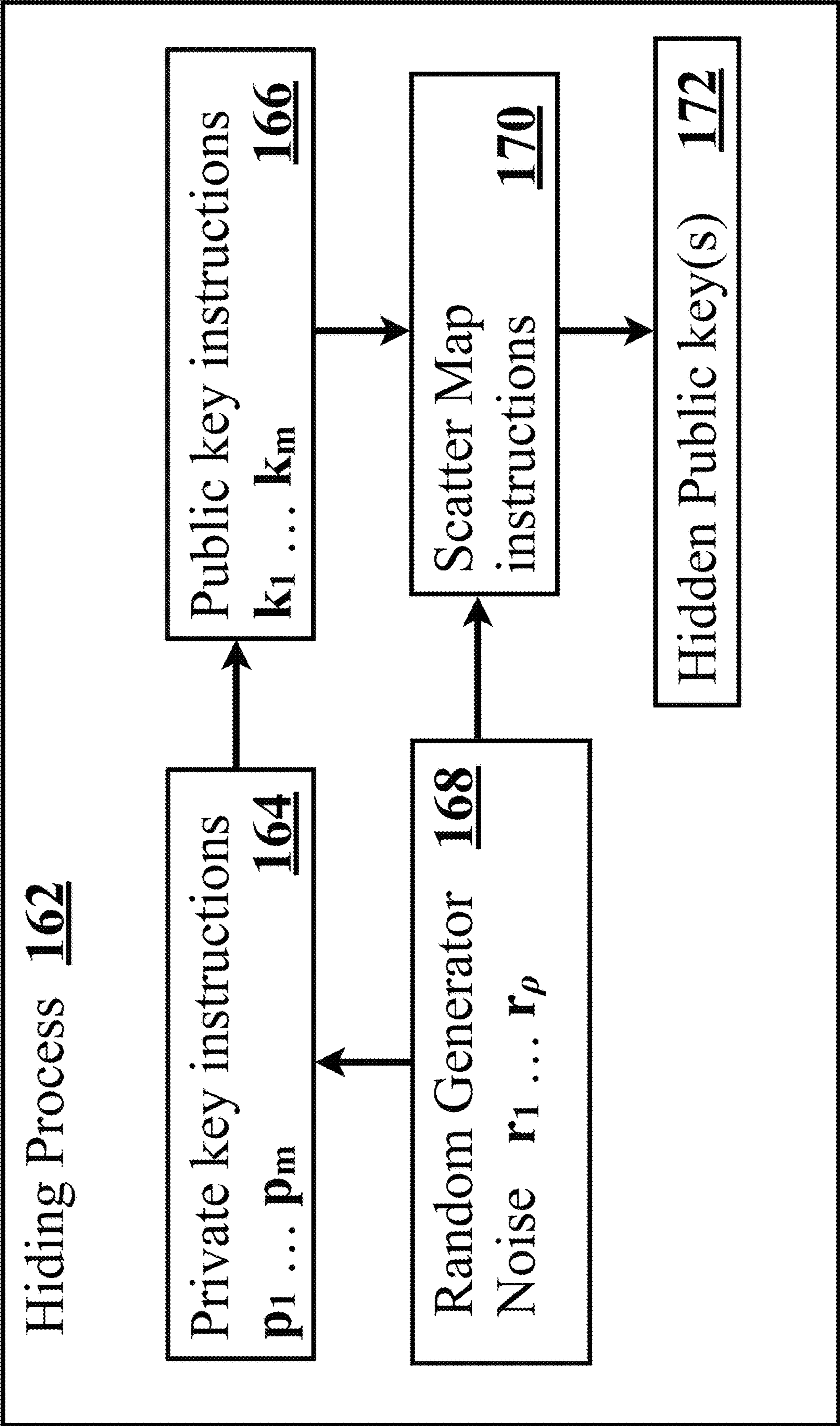


FIG. 1E

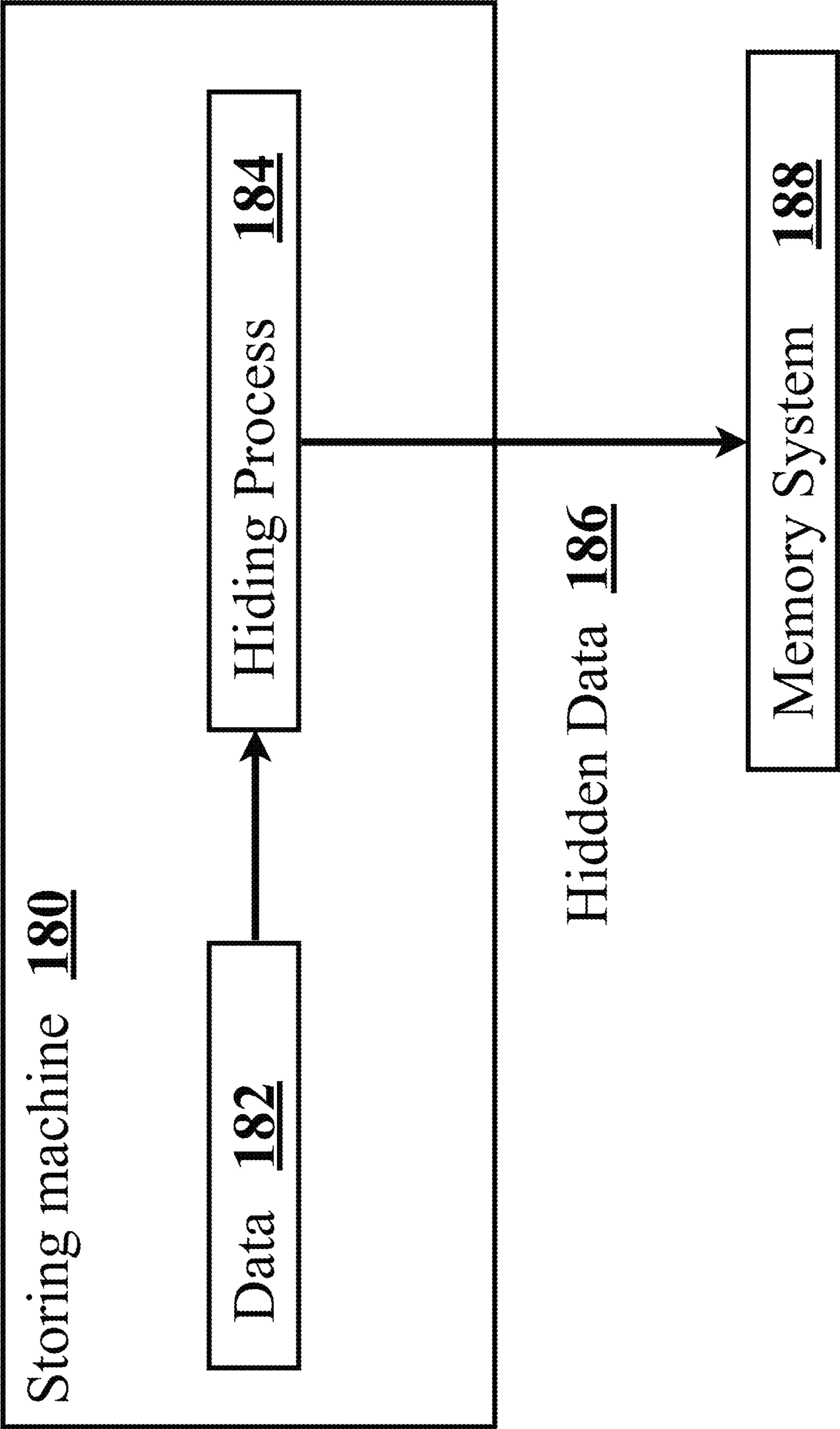
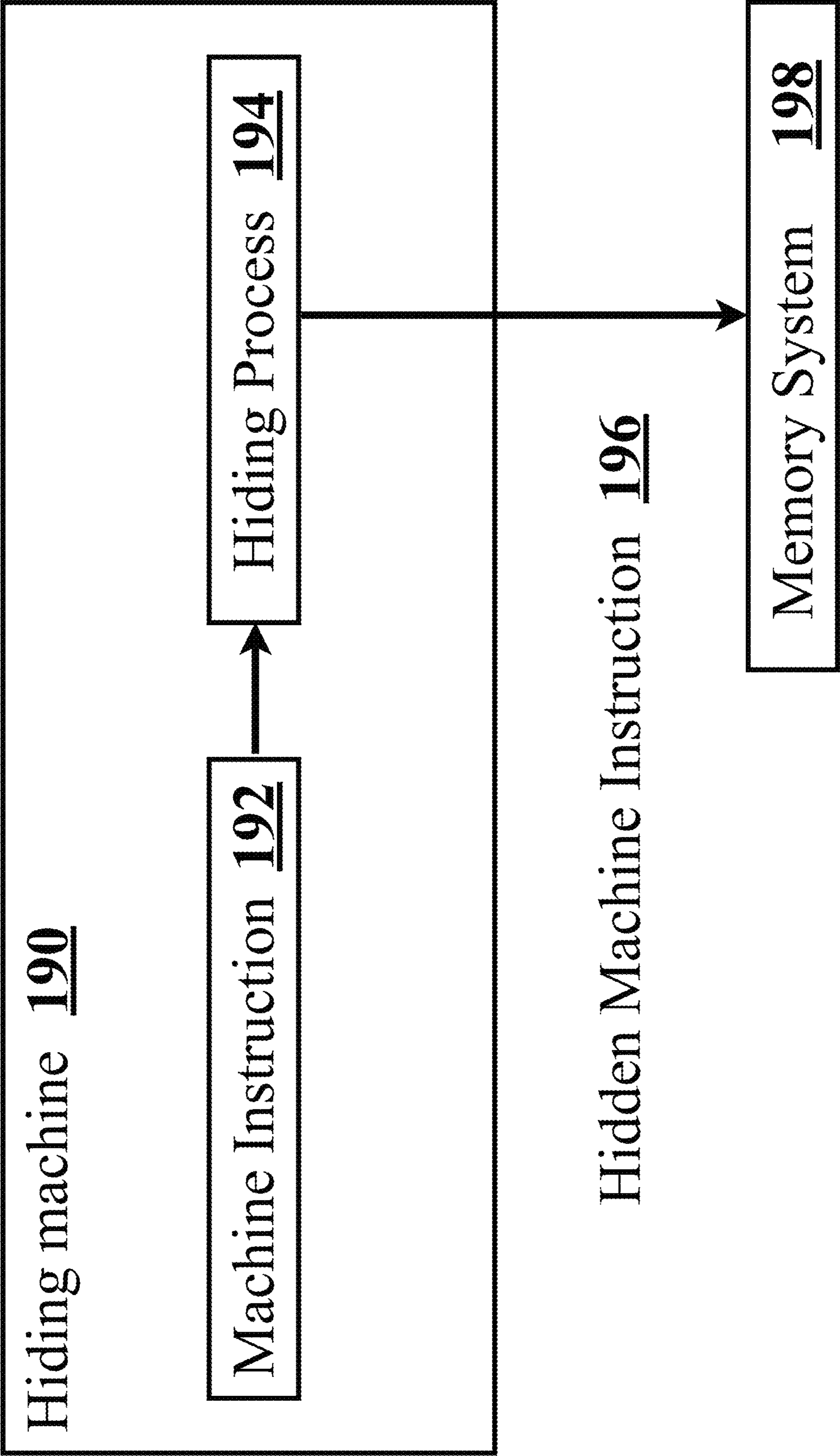
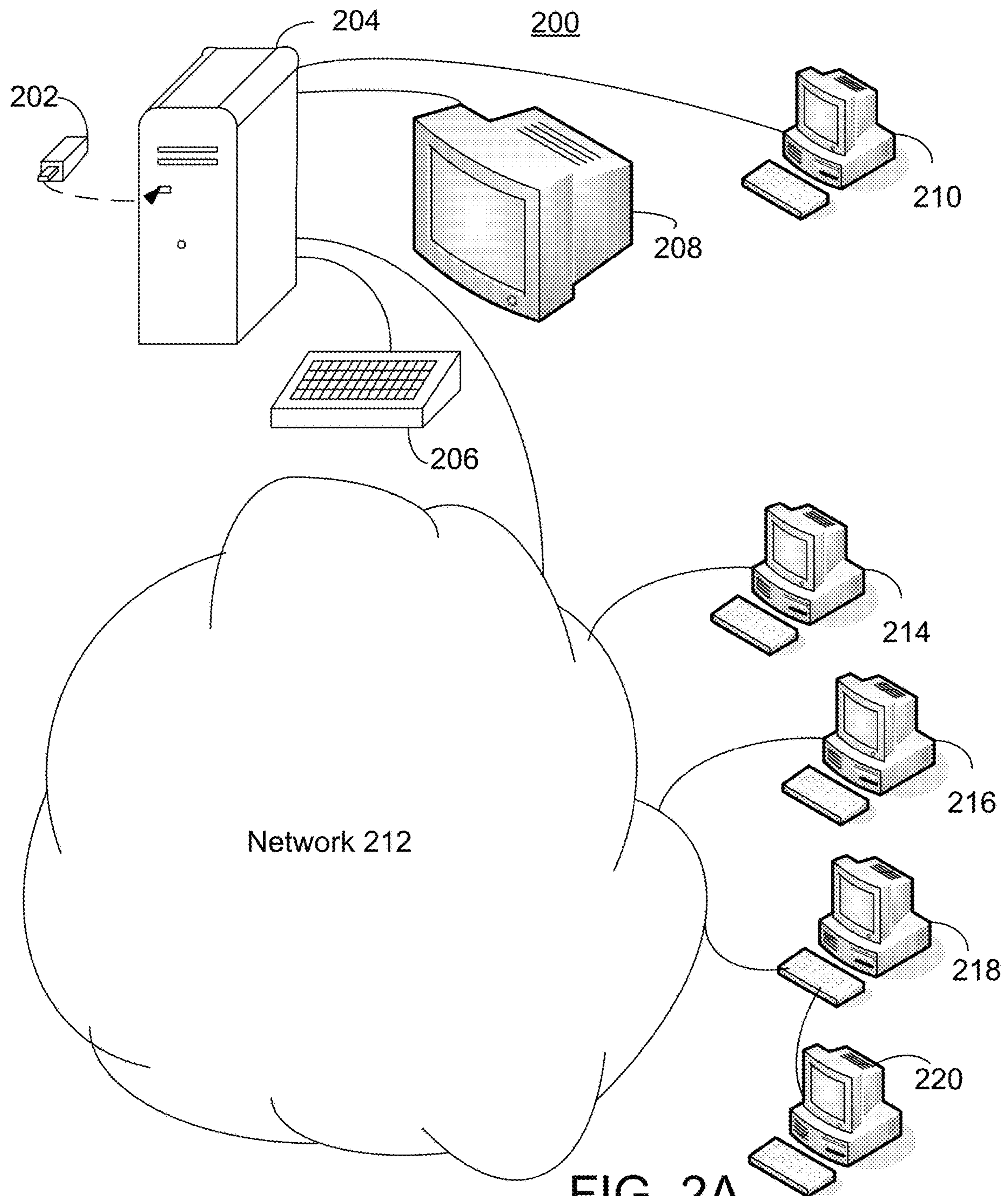


FIG. 1F





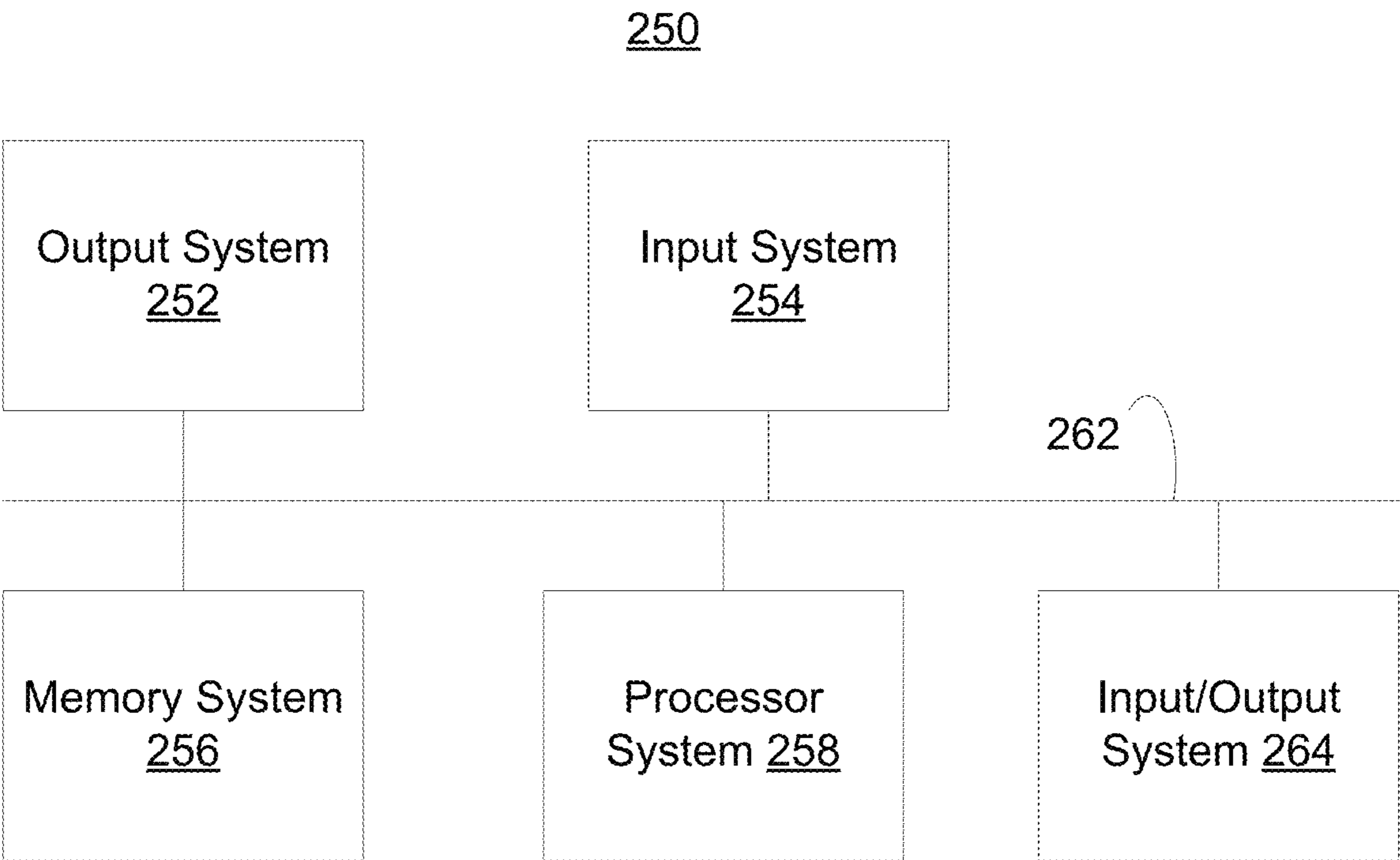


FIG. 2B

FIG. 3A

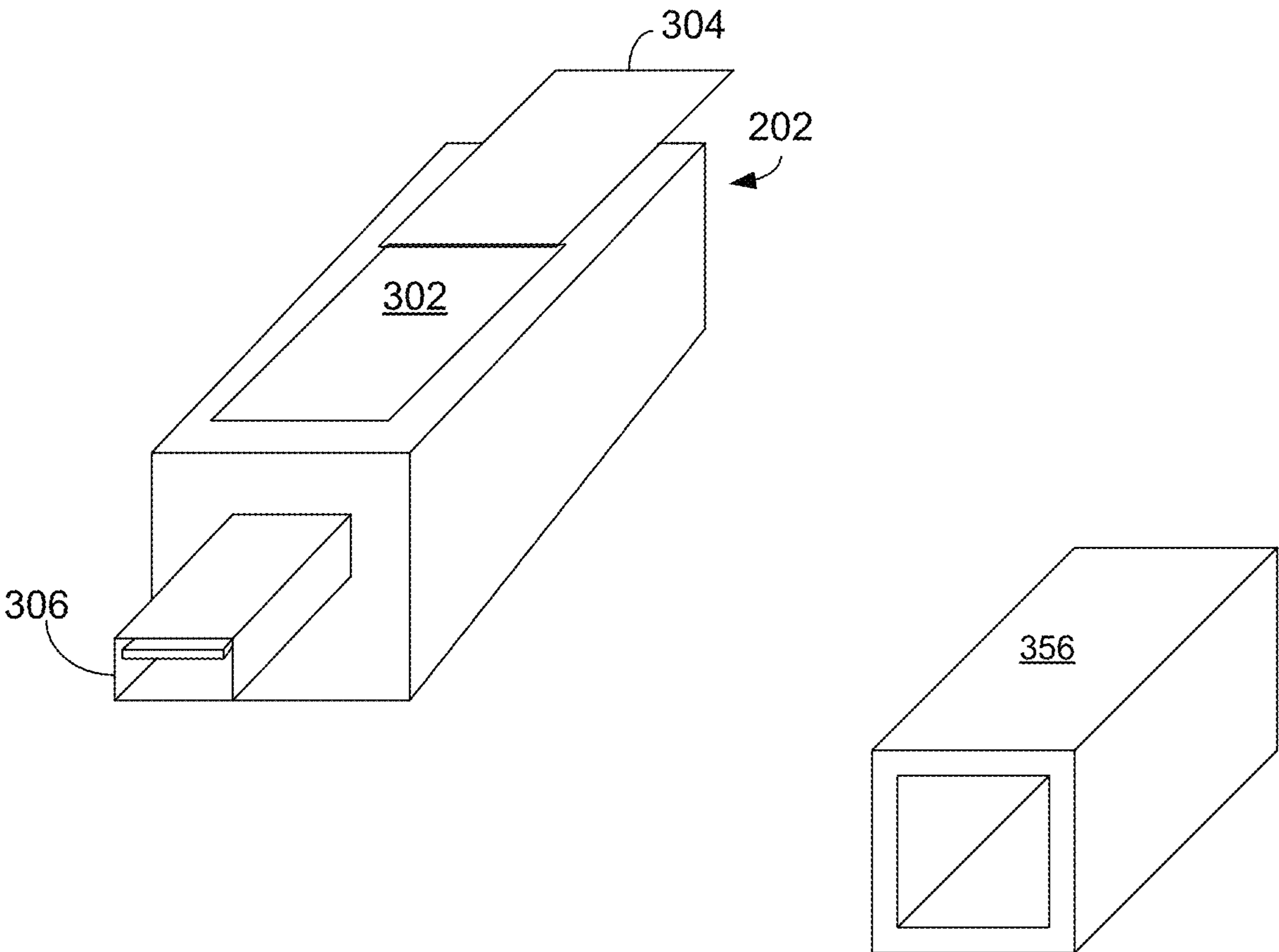


FIG. 3B

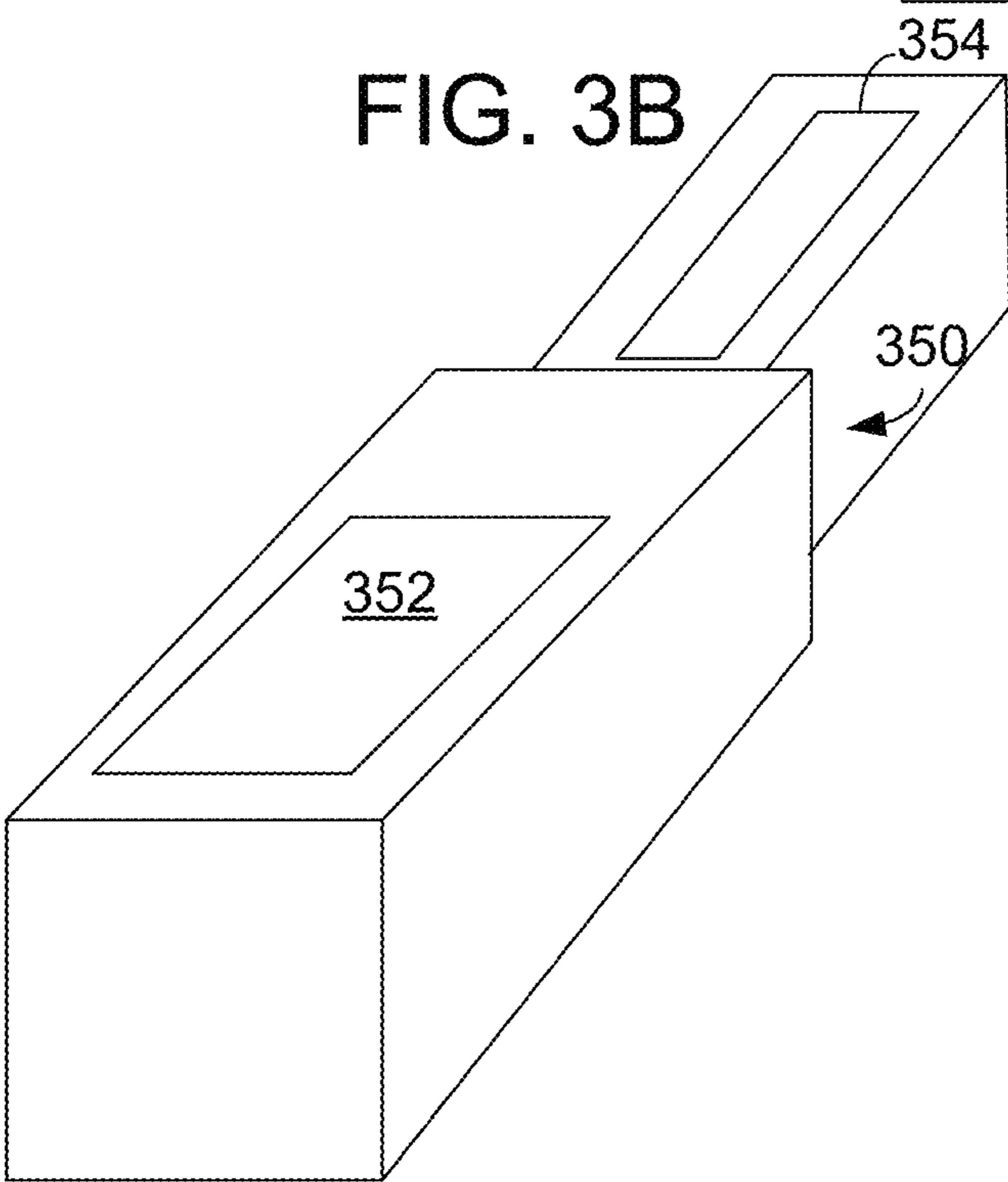


FIG. 4

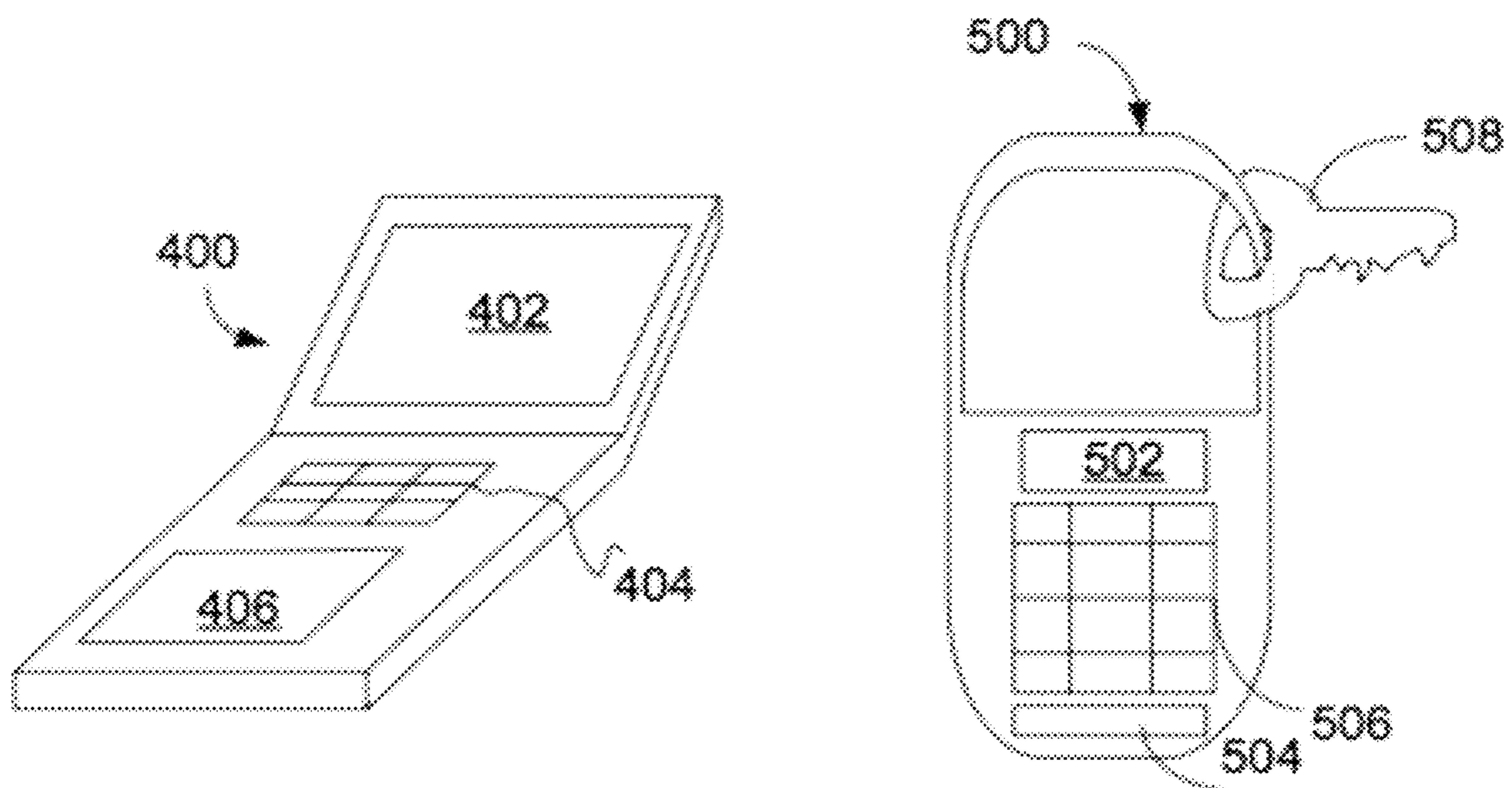
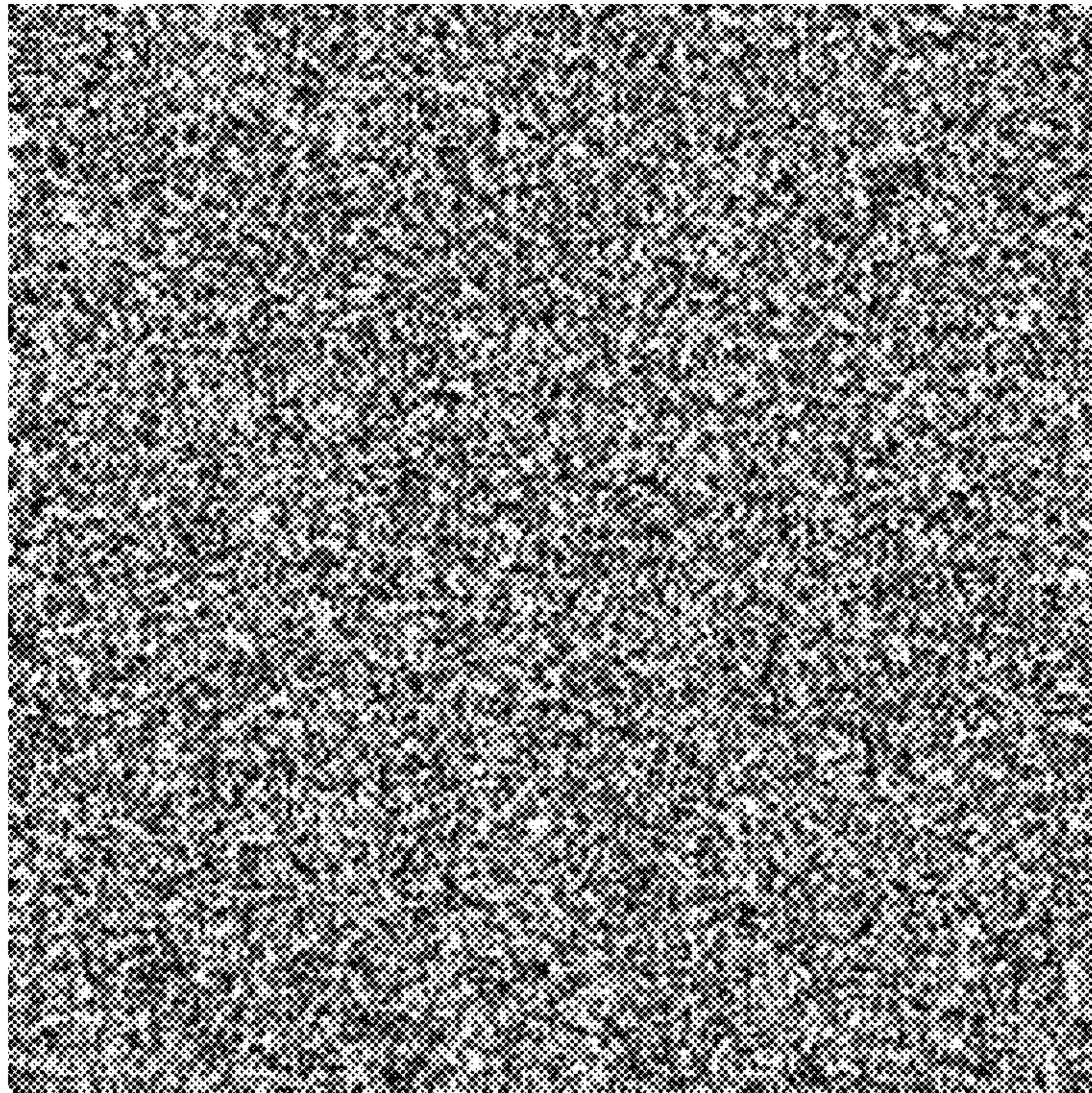
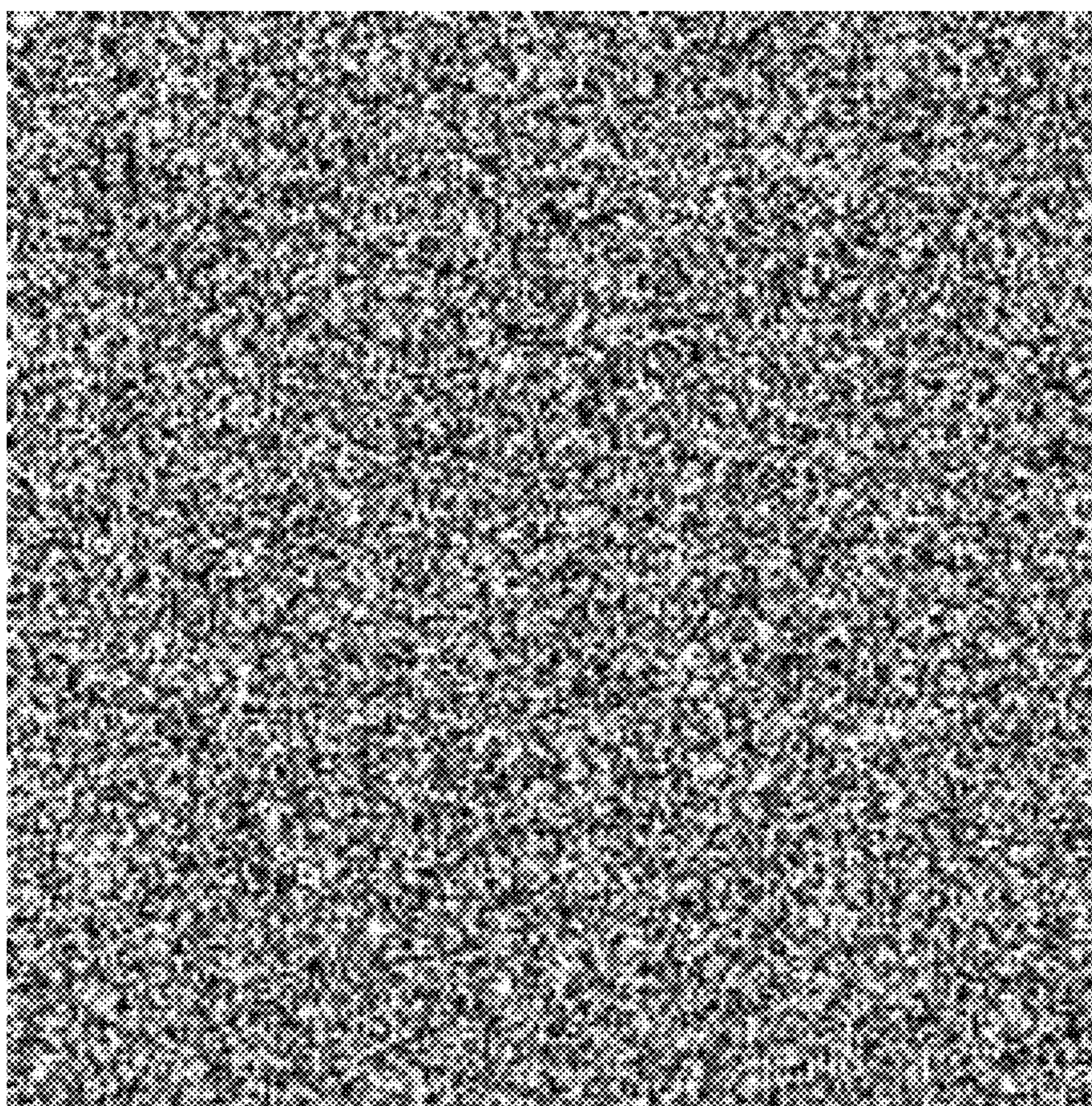


FIG. 5



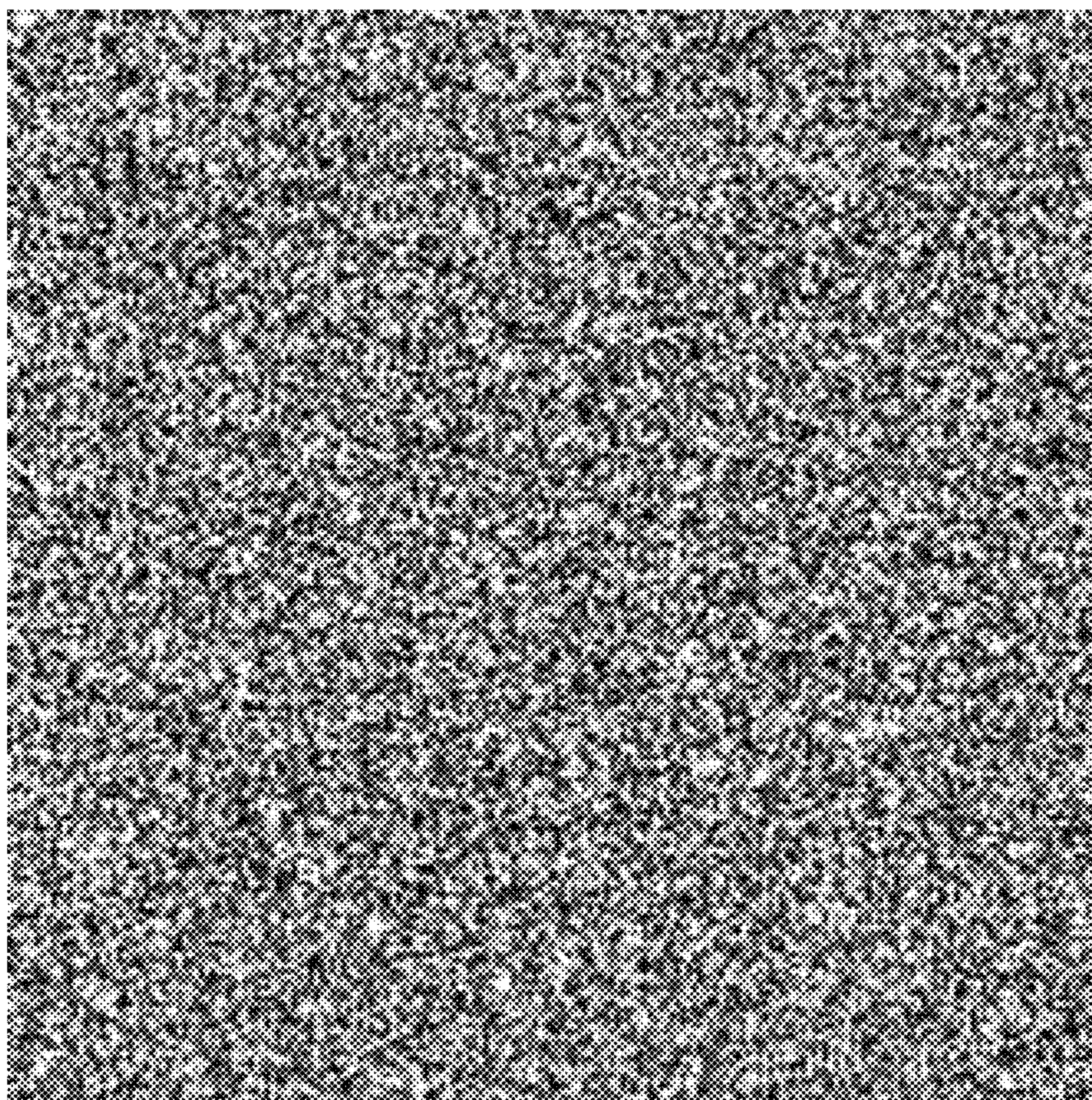
Hidden Key

FIG. 6A



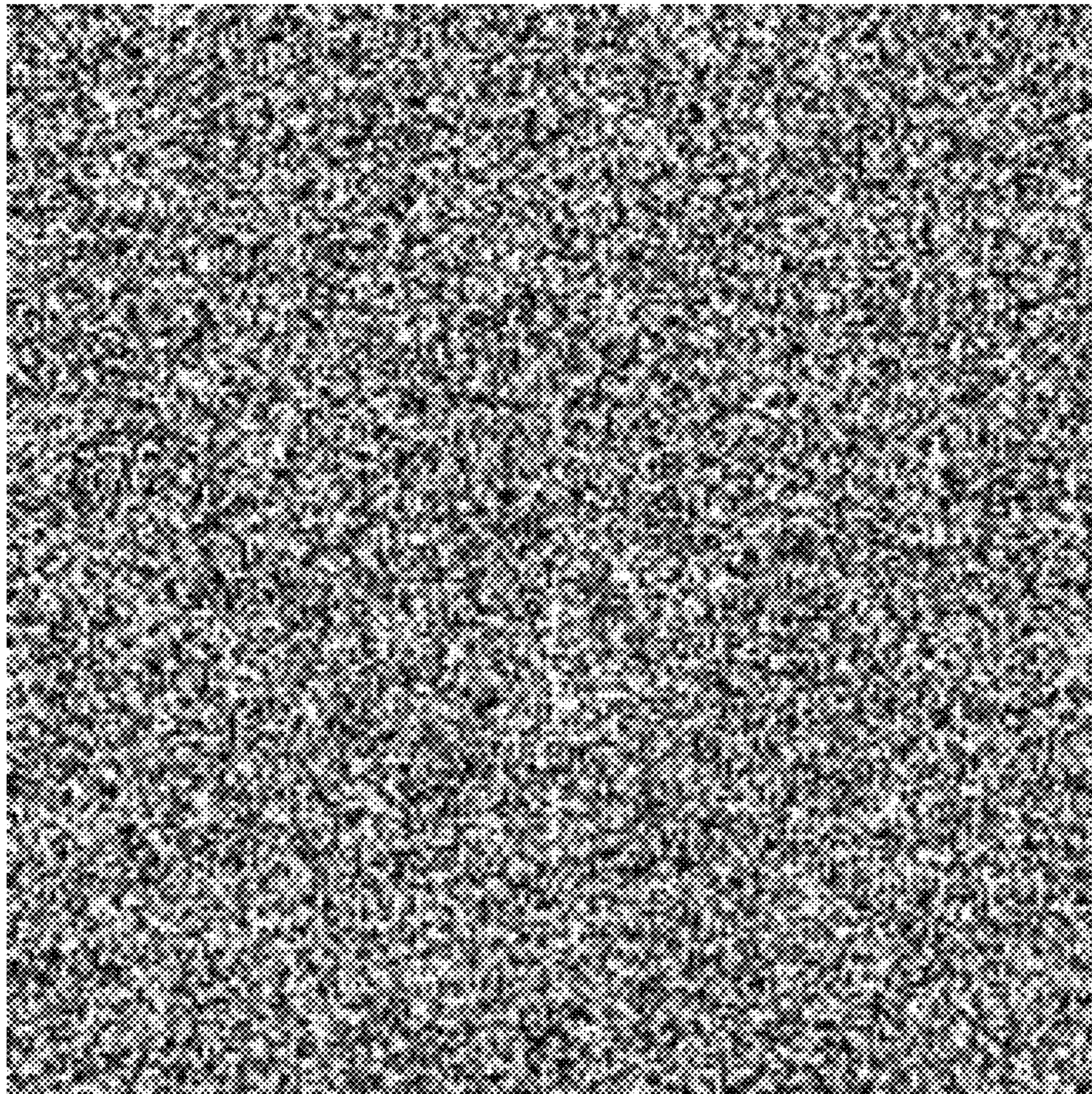
Hidden Data

FIG. 6B



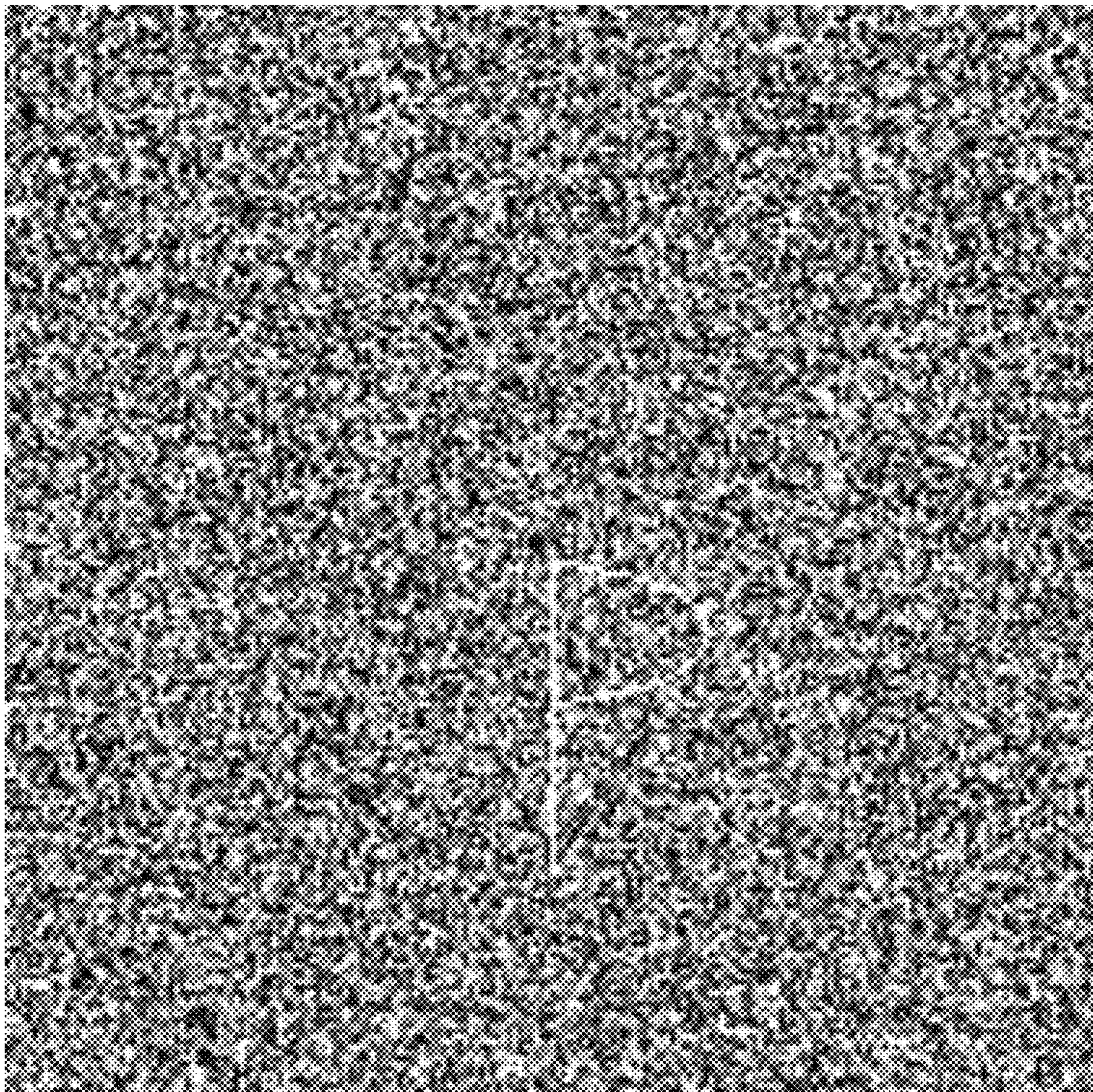
Hidden Instruction

FIG. 7



Somewhat Biased

FIG. 8



More Biased

FIG. 9A

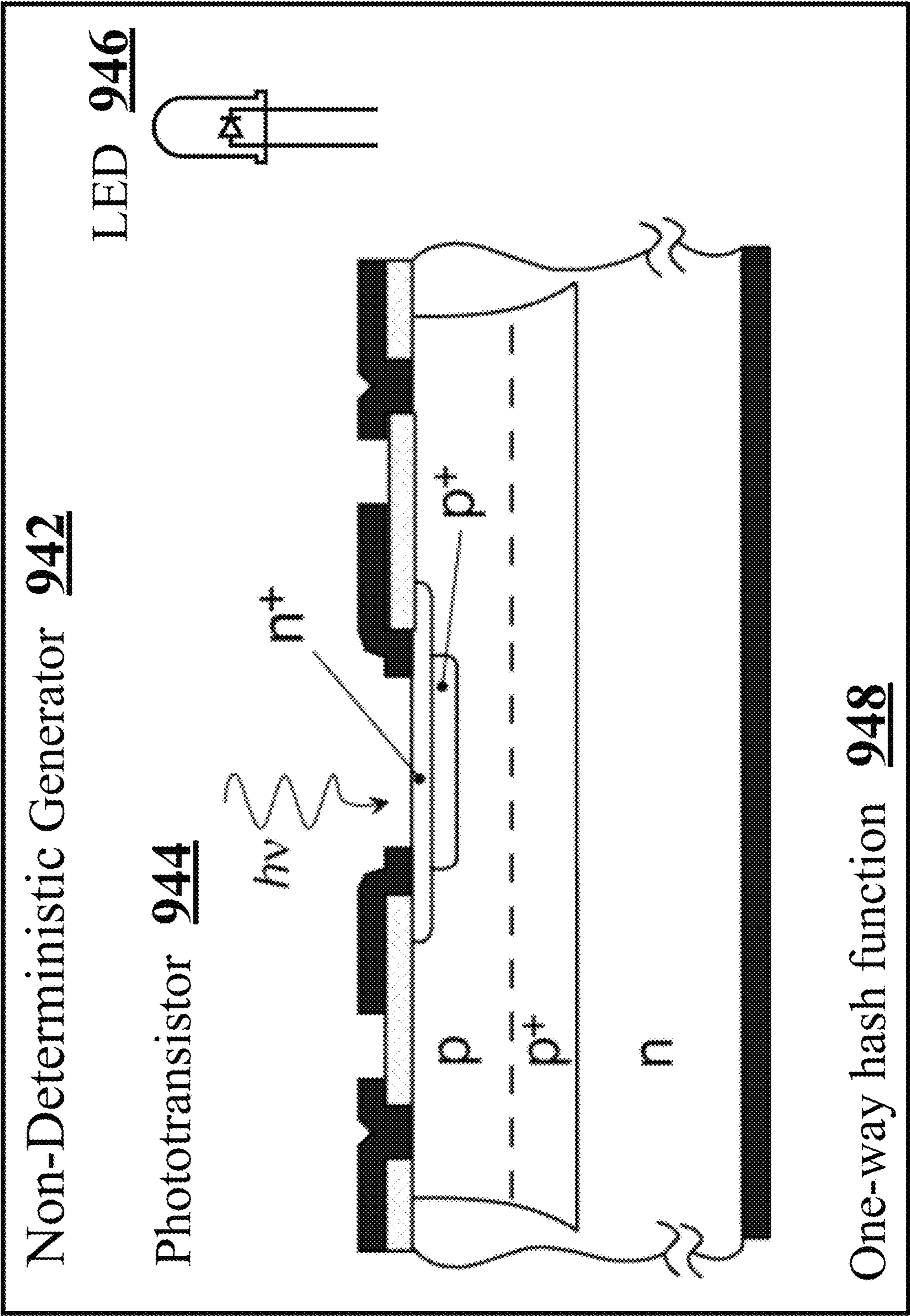


FIG. 9B

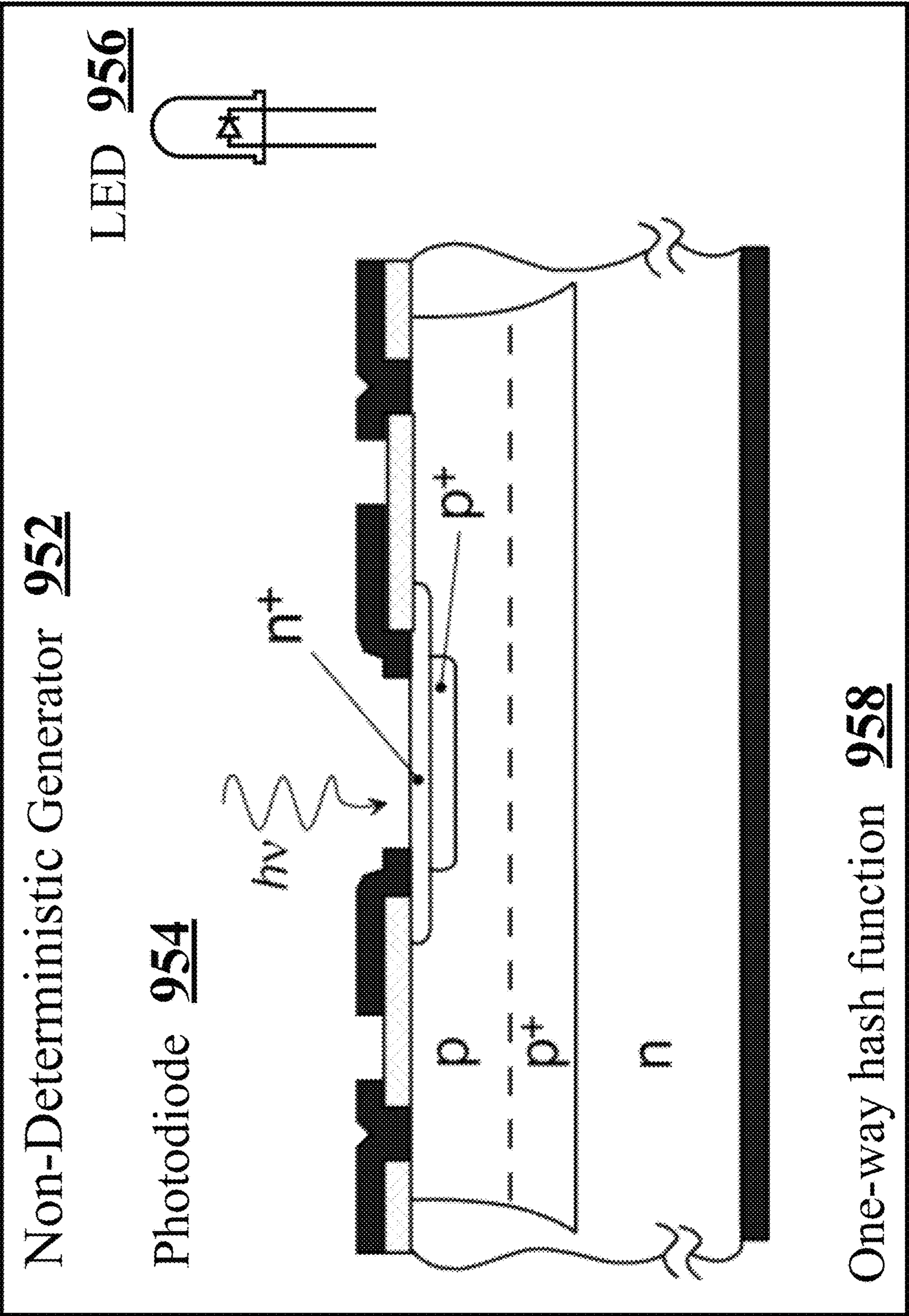


FIG. 9C

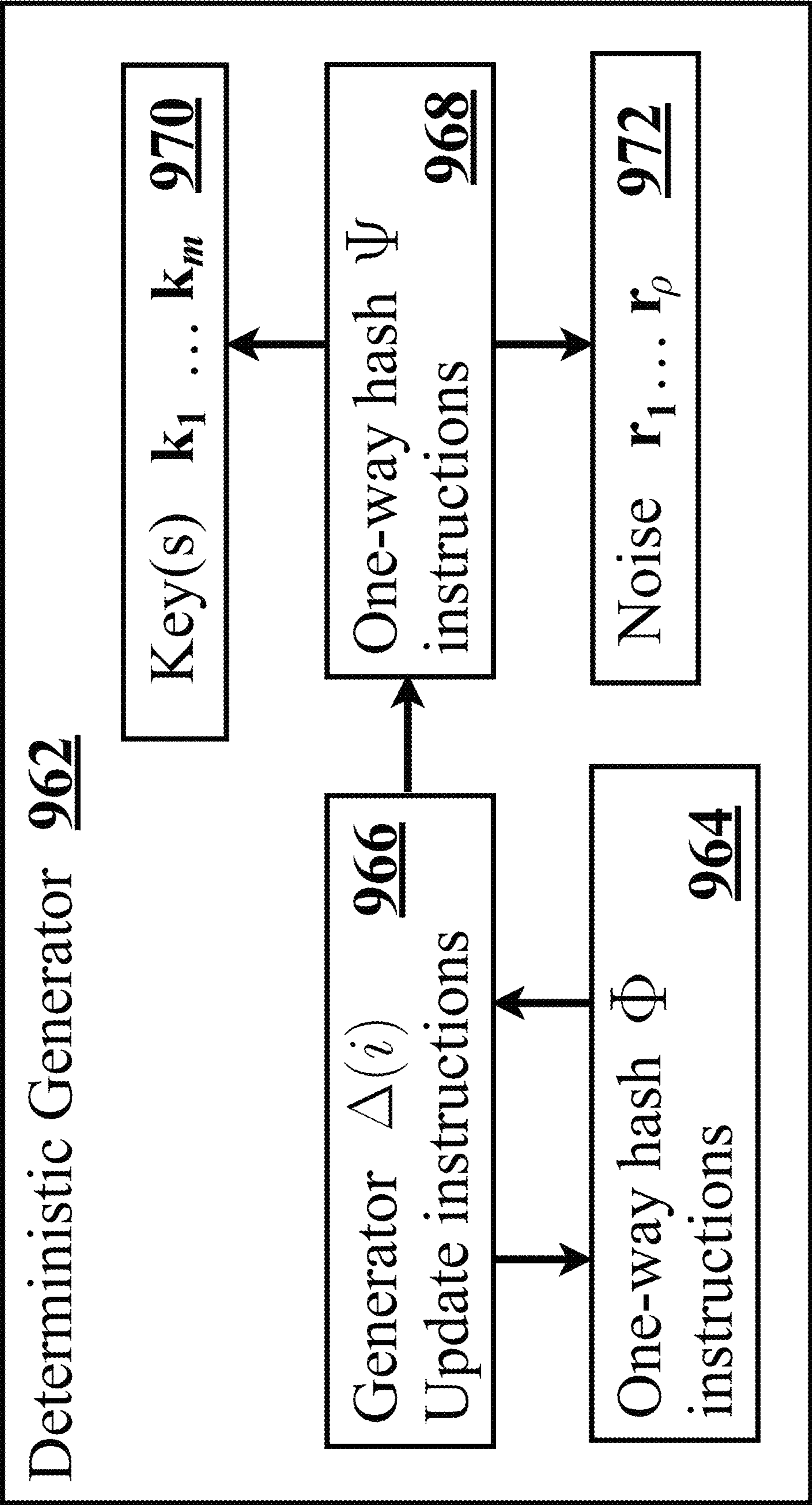


FIG. 9D

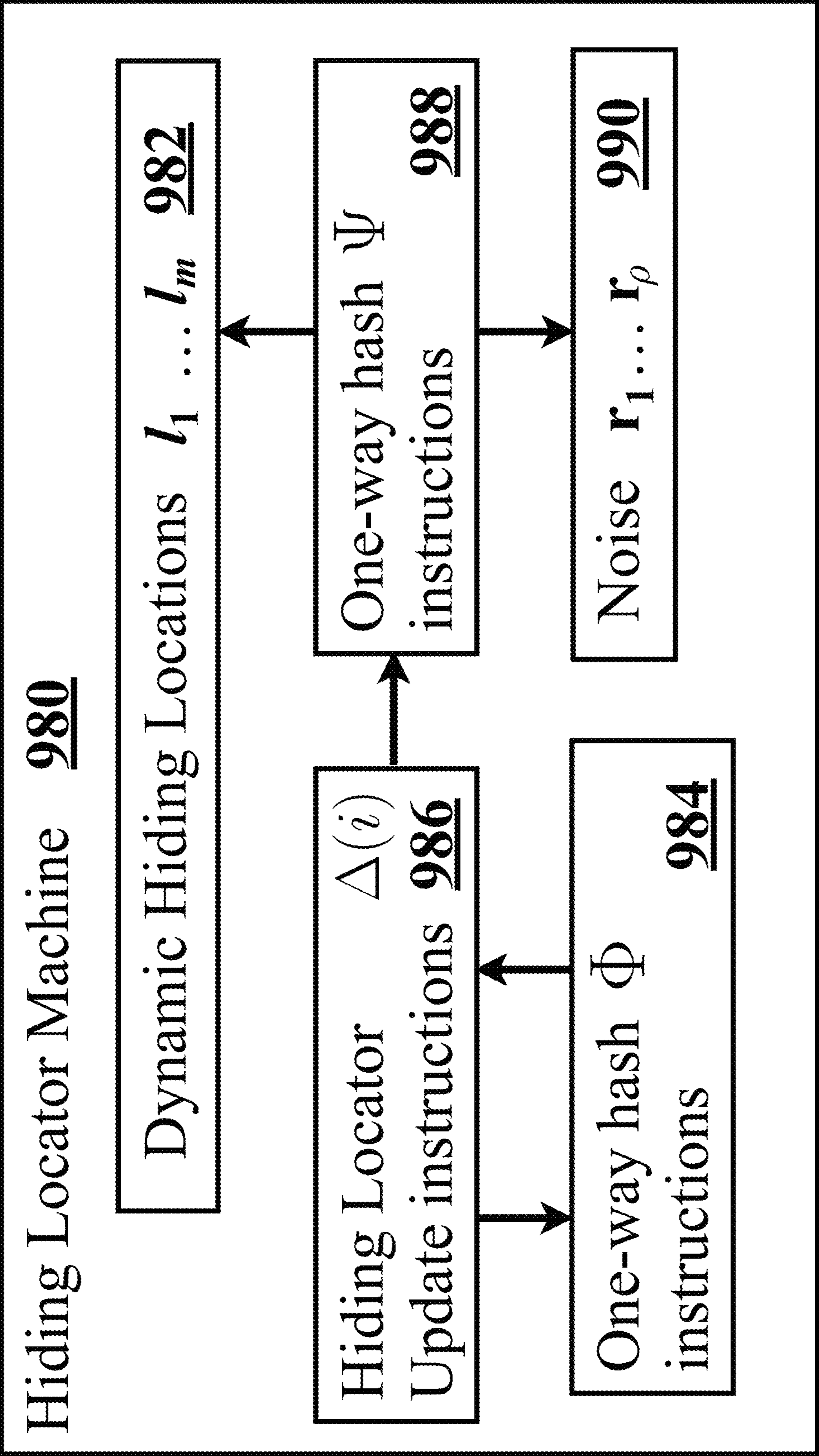


FIG. 10

Light Emitting Diode 1002

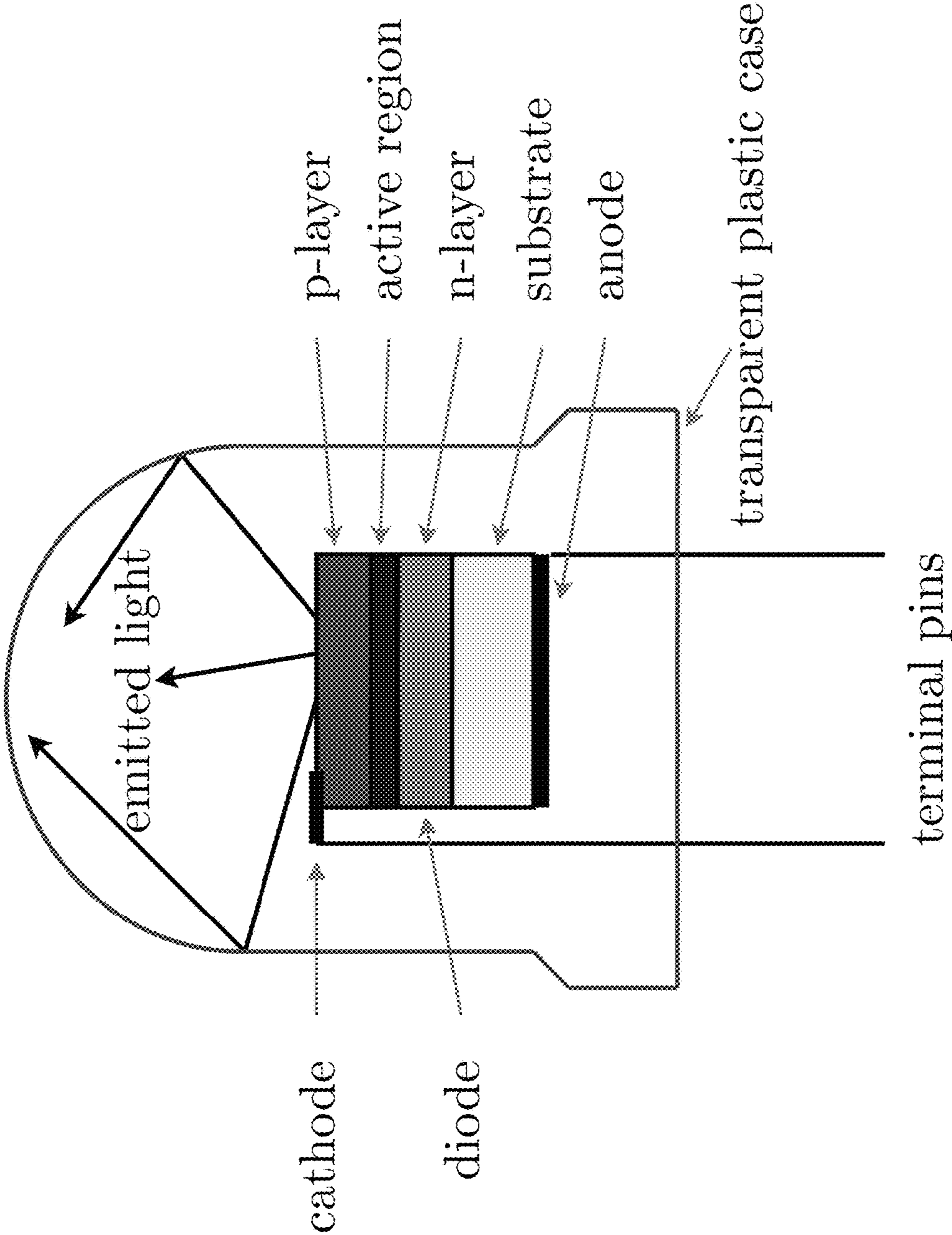


FIG. 11

Hiding 128 Bits

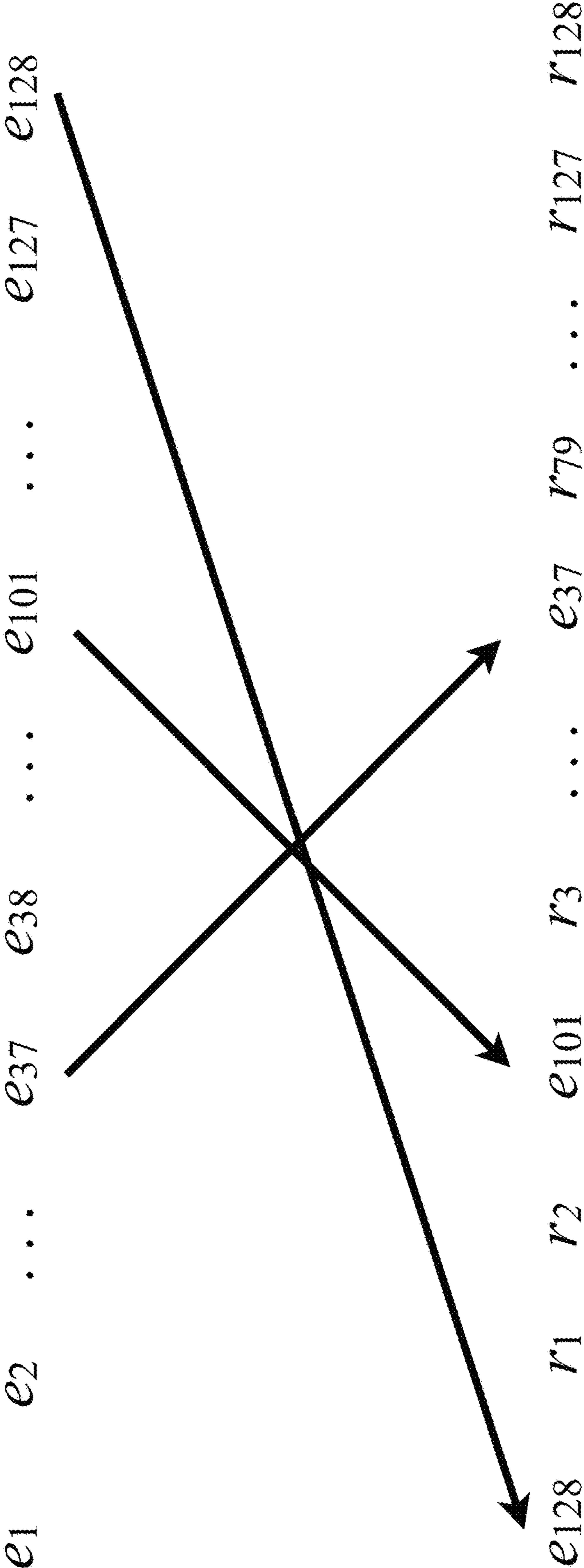


FIG. 12

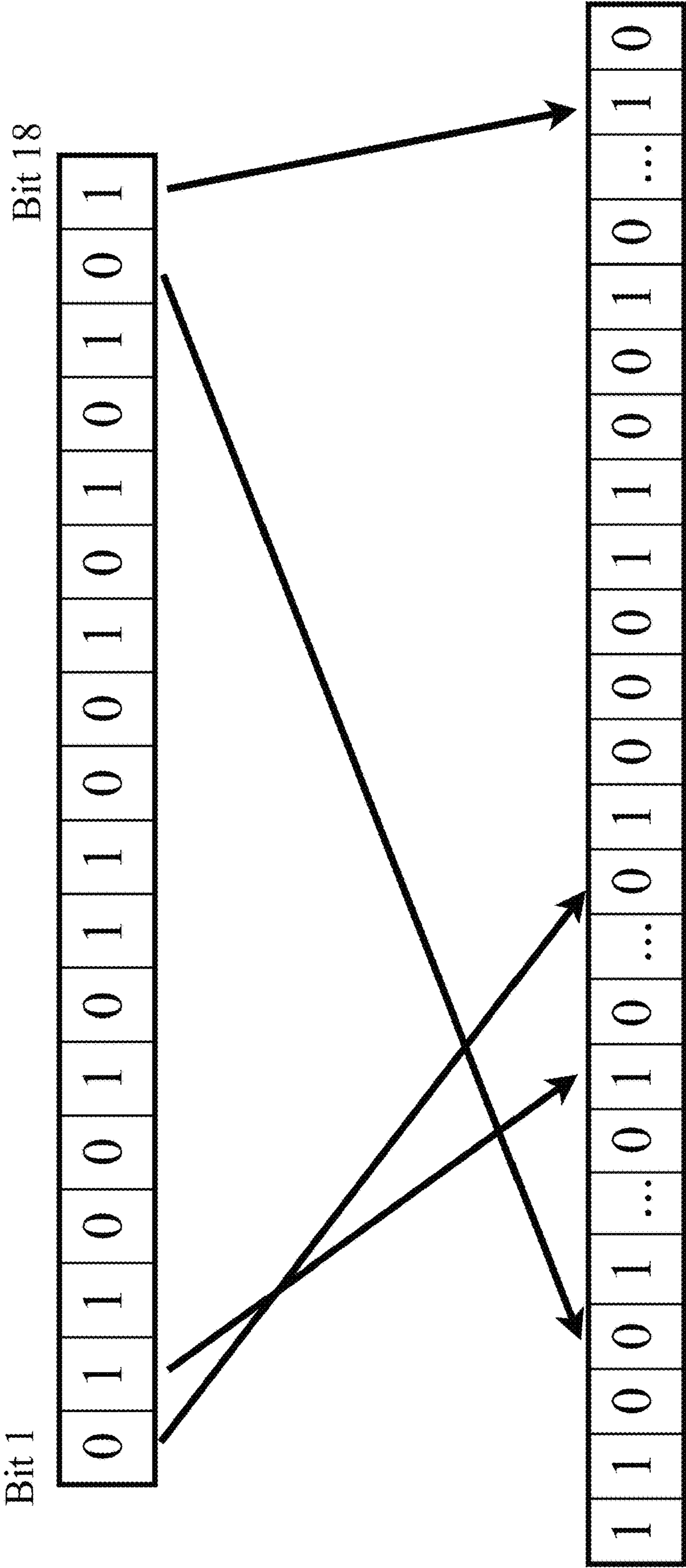


FIG. 13

Eve sees (m, n) scatter $\mathcal{S} = (s_1 \dots s_n)$

Event Name	Event Probability	Event Description
$B_{i,j}$	2^{-m}	$k_1 k_2 \dots k_m$ is the i th data in $E_{j,m}$
R_i	$\binom{\rho}{i} 2^{-\rho}$	$\eta_1(r_1 r_2 \dots r_\rho) = i$
A_i	$\binom{n}{i} 2^{-n}$	$\eta_1(s_1 \dots s_n) = i$

FIG. 14

Sum Changed to a Product

```
Fiskesoftware@Michaels-MacBook-Air C_program % ./ADD
```

```
2 3 5
```

```
result = 10. address of addition = 1 0 1 0 1 1 0 0 0 1 0 1 0 0 0 0
```

```
result = 30. address of multiply = 1 0 1 0 1 1 0 0 0 1 1 1 0 0 0 0
```

FIG. 15

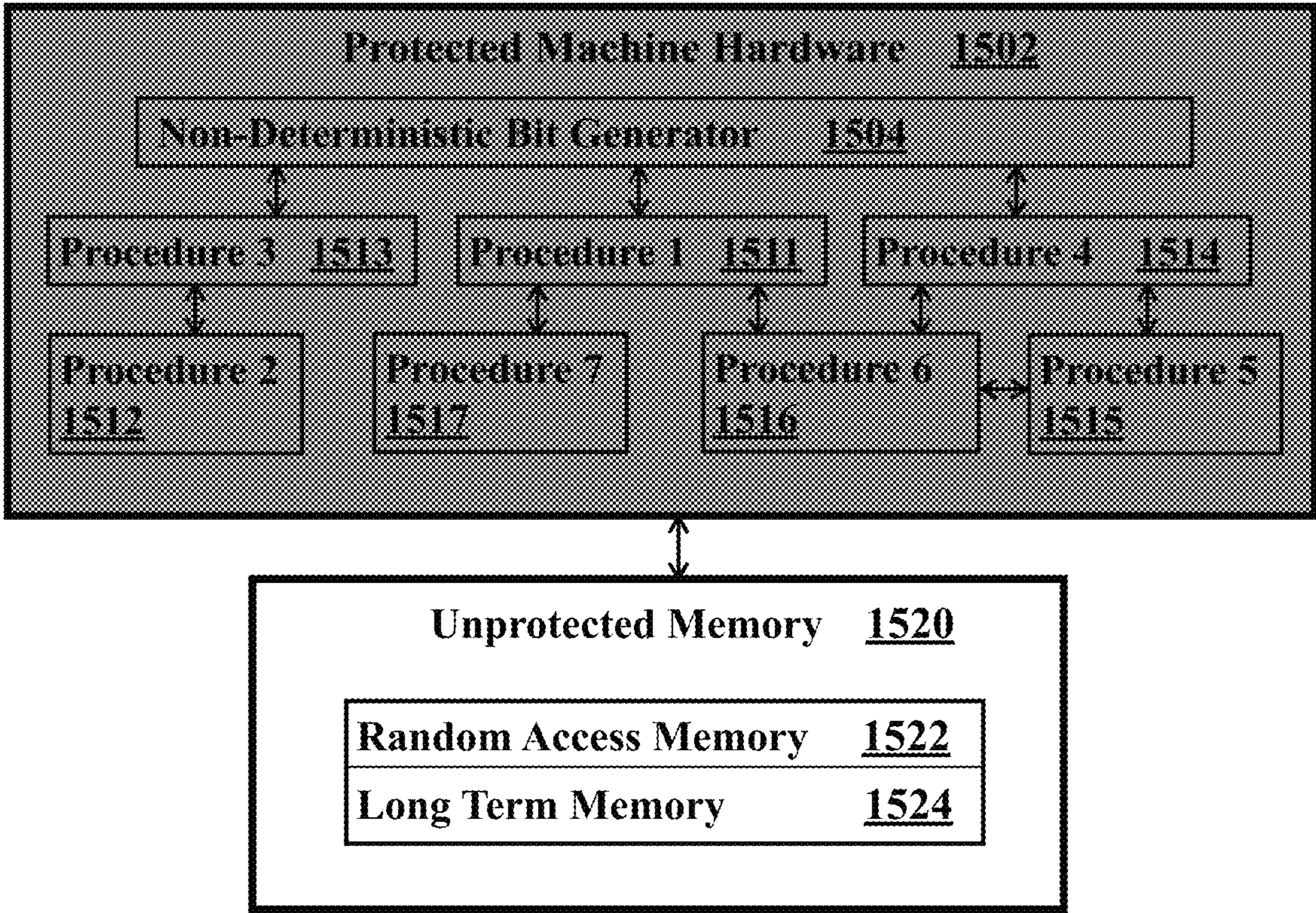


FIG. 16

Calculations of Ratio $\frac{f(n)}{e(n)} = \frac{n!}{2^n}$

```
Fiskesoftware@Michaels-MacBook-Air GMP_n_factorial_div_2_exp_n % ./GMP_factorial
```

```
64! / 2^64 = 6878554376797824332759832514784172552232874611144691171329550170898438
```

```
128! / 2^128 = 11332367464464471065863349006488644277280625343010794778403091336579  
083645024134334264308023333857802687981809757372916144651090017214050652063338777583  
85667111724615097045898438
```

```
256! / 2^256 = 74082588974169283522579680565007115387928390320509195914706556754384  
941302104765378423430920179541984663669976333356055019050227912063470578660810447331  
601133124358491010062441916092451587581953378663882678760062963910514450693490183724  
055424383855519204691940728625091357228482291784930107009912779115392754853359057248  
826533604364777534456263044570497347575300439860357681112112012383036883933495886367  
38189961761236190795898438
```

FIG. 17

Instruction Order

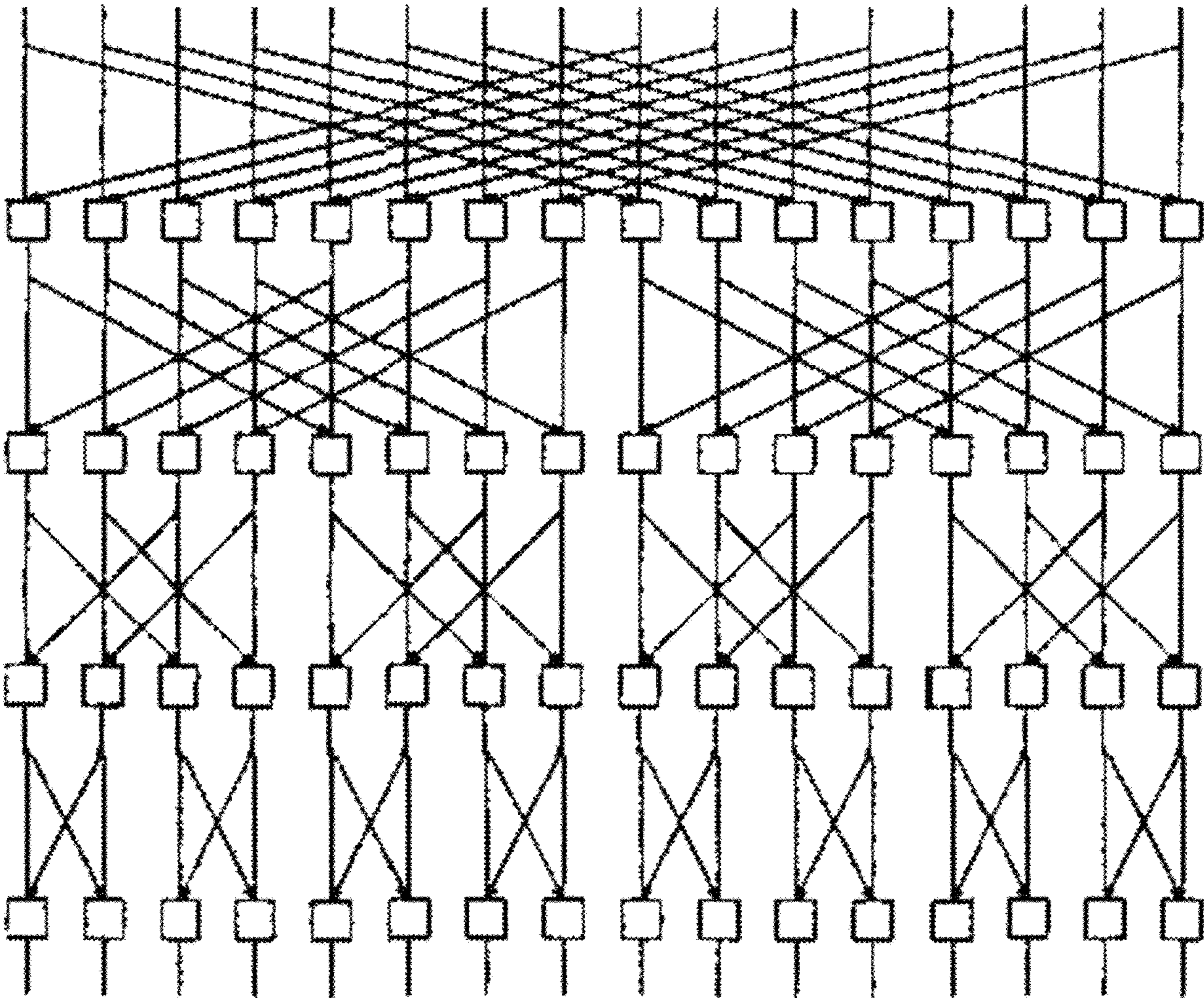
Block	Instruction	Instruction	Test $r \leq c$	Gamma
Index	Name	Type		Index
1	S_{j_1}	Valid	True	$\gamma(j_1)$
2	D_{r_1}	Dummy	False	$\gamma(r_1)$
...				
$\gamma^{-1}(1)$	S_1	Valid	True	$\gamma(1)$
...				
b	$D_{r_{k-1}}$	Dummy	False	$\gamma(r_{k-1})$

$j_i = \gamma^{-1}(i)$ and $r_i = \gamma^{-1}(i)$.

FIG. 18

16 Bit Butterfly Network

Input



Output

1

HIDING STABLE MACHINE INSTRUCTIONS
IN NOISE

1 RELATED APPLICATIONS

This application claims priority benefit of U.S. Provisional Patent Application Ser. No. 62/085,338, entitled “Hiding Data Transmissions in Random Noise”, filed Nov. 28, 2014, which is incorporated herein by reference; this application claims priority benefit of U.S. Provisional Patent Application Ser. No. 62/092,795, entitled “Hiding Data Transmissions in Random Noise”, filed Dec. 16, 2014, which is incorporated herein by reference; this application claims priority benefit of U.S. Non-provisional patent application Ser. No. 14/953,300, entitled “Hiding Information in Noise”, filed Nov. 28, 2015, which is incorporated herein by reference. This application claims priority benefit of U.S. Non-provisional patent Ser. No. 10,360,395, entitled “Hiding Information in Noise”, issued Jul. 23, 2019, which is incorporated herein by reference. This application claims priority benefit of U.S. Non-provisional patent Ser. No. 10,356,061, entitled “Hiding a Public Key Exchange in Noise”, issued Jul. 16, 2019, which is incorporated herein by reference.

This application is a continuation-in-part of U.S. Non-provisional Patent Application with Ser. No. 16/442,455, entitled “Dynamically Hiding Information in Noise”, filed Jun. 15, 2019. This application claims priority benefit of U.S. Provisional Patent Application Ser. No. 63/210,500, entitled “Hiding Stable Machine Instructions in Noise”, filed Jun. 15, 2021, which is incorporated herein by reference.

2 BACKGROUND—FIELD OF INVENTION

The present invention relates broadly to protecting the privacy of computer and machine instructions, information and devices. The processes and device are generally used to maintain the privacy of information transmitted through communication and transmission systems, and maintain the privacy of the computer instructions. For example, the hiding processes may be used to protect the metadata of a phone call; in some embodiments, the phone call may be transmitted via voice over IP (internet protocol) with a mobile phone. These processes and devices also may be used to hide passive data stored on a computer or another physical device such as a tape drive. In some embodiments, symmetric cryptographic methods and machines are also used to supplement the hiding process.

In an embodiment, the information (data) is hidden by a sending agent, called Alice. Alice transmits the hidden data to a receiving agent, called Bob. The receiving agent, Bob, applies an extraction process or device. The output of this extraction process or device is the same information (data) that Alice gathered before hiding and sending it. Eve is the name of the agent who is attempting to obtain the information or data. One of Alice and Bob’s primary objectives is to assure that Eve cannot capture the private information that was hidden and transmitted between them.

In another embodiment, Alice desires to hide data and securely store the hidden data somewhere and retrieve it and access the hidden data at a later time. The output of this extraction process or device is the same information (data) that Alice gathered before hiding and storing it.

In some embodiments of our invention we describe computing machine that is structurally stable to small changes made to its program instructions. Our procedures use quantum randomness to build unpredictable stable instructions

2

and use quantum randomness to generate noise such that the program instructions are hidden in the noise. In some embodiments, our procedures can execute just before running a program so that the computing task can be performed with a different representation of its program instructions during each run. In some embodiments, our invention expands the engineering method of stability to a cryptographically stable machine with an adversary; our invention describes cryptographically stable computing machines that are resistant to malware sabotage.

3 BACKGROUND—PRIOR ART

The subject matter discussed in this background section should not be assumed to be prior art merely as a result of its mention in the background section. Similarly, a problem mentioned in the background section or associated with the subject matter of the background section should not be assumed to have been previously recognized in the prior art. The subject matter in the “Summary and some Advantages of Invention” section represents different approaches, which in and of themselves may also be inventions, and various problems, which may have been first recognized by the inventor.

In information security, a fundamental problem is for a sender, Alice, to securely transmit a message M to a receiver, Bob, so that the adversary, Eve, receives no information about the message. In Shannon’s seminal paper [2], his model assumes that Eve has complete access to a public, noiseless channel: Eve sees an identical copy of ciphertext C that Bob receives, where $C(M, K)$ is a function of message M lying in message space \mathcal{M} and secret key K lying in key space \mathcal{K} .

In this specification, the symbol P will express a probability. The expression $P(E)$ is the probability that event E occurs and it satisfies $0 \leq P(E) \leq 1$. For example, suppose the sample space is the 6 faces of die and E is the event of rolling a 1 or 5 with that die and each of the 6 faces is equally likely. Then

$$P(E) = \frac{2}{6} = \frac{1}{3}.$$

The conditional probability

$$P(A|B) = \frac{P(A \cap B)}{P(B)}.$$

$P(A \cap B)$ is the probability that event A occurs and also event B occurs. The conditional probability $P(A|B)$ expresses the probability that event A will occur, under the condition that someone knows event B already occurred. The expression that follows the symbol “|” represents the conditional event. Events A and B are independent if $P(A \cap B) = P(A) P(B)$.

Expressed in terms of conditional probabilities, Shannon [2] defined a cryptographic method to be perfectly secret if $P(M) = P(M|Eve \text{ sees ciphertext } C)$ for every cipher text C and for every message M in the message space \mathcal{M} . In other words, Eve has no more information about what the message M is after Eve sees ciphertext C pass through the public channel. Shannon showed for a noiseless, public channel that the entropy of the keyspace \mathcal{K} must be at least as large as the message space \mathcal{M} in order to achieve perfect secrecy.

Shannon's communication secrecy model [2] assumes that message sizes in the message space are finite and the same size. Shannon's model assumes that the transformations (encryption methods) on the message space are invertible and map a message of one size to the same size. Shannon's model assumes that the transformation applied to the message is based on the key. In the prior art, there is no use of random noise that is independent of the message or the key. In the prior art, there is no notion of being able to send a hidden or encrypted message inside the random noise where Eve is not necessarily revealed the size of the message. In the prior art, there is no notion of using random noise to hide the secret channel and transmitting a key inside this channel that is indistinguishable from the noise.

Quantum cryptography was introduced by Weisner and eventually published by Bennett, Brassard, et al. [3, 4]. Quantum cryptography based on the uncertainty principle of quantum physics: by measuring one component of the polarization of a photon, Eve irreversibly loses her ability to measure the orthogonal component of the polarization. Unfortunately, this type of cryptography requires an expensive physical infrastructure that is challenging to implement over long distances [5, 6]. Furthermore, Alice and Bob still need a shared, authentication secret to successfully perform this quantum cryptography in order to assure that Eve cannot corrupt messages about the polarization bases, communicated on Alice and Bob's public channel.

4 BACKGROUND: The Malware Problem in Prior Art

Malware plays a critical role in breaching computer systems. Cybersecurity research has primarily focused on malware detection [7, 8, 9, 10]. It is unlikely that malware detection methods can solely provide an adequate solution to the malware problem: There does not exist a register machine algorithm that can detect all malware [11]. Furthermore, some recent malware implementations use NP problems to encrypt and hide the malware [13]. Overall, detection methods are currently up against fundamental limits in theoretical computer science [14].

During a 20 year study, the DARPA program, CRASH [15], compared the number of lines of source code in security software versus malware. Security code grew from about 10,000 to 10 million lines; while malware code was almost constant at 125 lines. This study found that an almost constant number of lines of malware code can hijack a computer program, independent of the security software's size.

The instability of register machine [16, 17] computers enables malware to sabotage the purpose of a computer program, by making small changes to one or more instructions in an original, uninfected program. Programming languages such as C, Java, Lisp and Python rely upon branching instructions. After a branching instruction of a register machine program has been sabotaged, the malware can start executing. Even if there is a routine (in hardware and/or software) to verify that the program is executing properly, this verification routine may never execute. The sequential execution of unstable register machine instructions cripples the program from protecting itself.

5 Advantages and Novelty Over Prior Art

The prior art in mathematical research, computer science, and electrical engineering has not attempted to design malware resistant computation based on structural stability. For

over 80 years, dynamical systems has extensively studied structural stability [18, 19, 20, 21] on phase spaces [22], containing an uncountable number of states. (A space X is uncountable if X contains an infinite number of states and there does not exist a one-to-one correspondence between X and the natural numbers $N=\{0, 1, 2, 3, 4, 5, 6, 7, \dots\}$.)

During the execution of a register machine [16, 17], the machine's state at any moment lies in a discrete space, containing a countable number of states. (A space X is countable if there exists a one-to-one correspondence between X and N .) Based on dynamical systems and information theory, a novelty of our invention(s) introduces mathematical and computational tools to build a structurally stable sequential machine. Structurally stable means the sequential machine is resistant to small changes to the program instructions. Our invention(s) are inspired by the Red Queen hypothesis in evolutionary biology: organisms evolve using robustness, unpredictability and variability to hinder infection from parasites. In some embodiments, a novel part of our invention(s) expands the engineering method of stability to a cryptographically stable machine with an adversary; our invention describes cryptographically stable computing machines that are resistant to malware sabotage.

6 SUMMARY AND SOME ADVANTAGES OF USING NOISE

In some parts of the prior art, conventional wisdom believes that hiding data in the open cannot provide adequate information security. The invention(s), described herein, demonstrate that our process of hiding data inside noise or hiding machine instructions inside noise can be quite effective. A process for hiding machine instructions inside of random noise is demonstrated and described. A process for hiding data inside of random noise is also demonstrated and described. In some embodiments, the data hidden is a key. In some embodiments, the data hidden is a public key. In some embodiments, the data hidden is encrypted data. In some embodiments, the data hidden is encrypted data that was first encrypted by a block cipher. In some embodiments, the instructions hidden are hidden instructions that were first encrypted by a block cipher. In some embodiments, the data hidden is encrypted data that was first encrypted by a stream cipher. In some embodiments, the hidden data may be hidden metadata that is associated with the π CP/IP infrastructure [1] used to transmit information.

The invention(s) described herein are not bound to Shannon's limitations [2] because they use noise, rather than seek to eliminate noise. When the data generation and random noise have the same probability distribution or both have a uniform probability distribution, and the key size is fixed, the security of the key transmission can be made arbitrarily close to perfect secrecy—where arbitrarily close is defined in section 10.11—by increasing the noise size. The processes, devices and machines described herein are practical; they can be implemented with current π CP/IP infrastructure acting as a transmission medium and a random noise generator providing the random noise and key generation.

Overall, our invention(s) that hide machine instructions (opcodes and operands), data and keys inside random noise exhibits the following favorable security properties.

The hiding process is $O(n)$.

For a fixed key size m bits and $\rho=n-m$ bits of random noise, as $\rho \rightarrow \infty$, the security of the hidden data can be made arbitrarily close to perfect secrecy. In some

5

applications, the key size can also be kept secret and is not revealed to Eve. From the binomial distribution, the closeness to perfect secrecy can be efficiently computed.

The scatter map a can be reused when both the key generation and noise generation have the same or uniform probability distribution and a new random key and new noise are created for each transmission. The reuse property enables a practical process of hiding data that is first encrypted by a block or stream cipher. The complexity of finding this hidden encrypted data can be substantially greater than the computational complexity of the underlying block or stream cipher. See section 10.14.

Our hiding process uses a noiseless, public channel, which means it can be implemented with our current Transmission Control Protocol/Internet Protocol internet infrastructure (TCP/IP). No expensive, physical infrastructure is needed to create noisy channels or transmit and maintain polarized photons, as is required by the prior art of quantum cryptography. Random noise generators are commercially feasible and inexpensive. A random noise generator that produces more than 10,000 random bits per second can be manufactured in high volume for less than three U.S. dollars per device.

Alice and Bob possess their sources of randomness. This system design decentralizes the security to each user. Decentralization helps eliminate potential single points of failure, and backdoors in the transmission medium that may be outside the inspection and control of Alice and Bob.

In an embodiment where Alice wishes to store her hidden data, the inventions described herein have an additional advantage over the prior art of cryptography. In the inventions described herein, Eve does not know the size of the hidden data or the number of hidden instructions in the hidden program. The noise size can be substantially larger than the data or program size so that hidden data or hidden program instructions could be a telephone number in one case and the whole sequence of DNA of human chromosome 4 in a second case of hiding. Typically in the prior art, an encryption preserves the file size or voice packet size.

In an embodiment Alice executes her machine instructions in Protected Machine Hardware and Eve is not able to sabotage Alice's machine instructions with malware.

6.1 Comparison to Prior Art Methods of Protecting Computation

Fully homomorphic encryption (FHE) is a method for protecting Alice's computations on data in the cloud. FHE DOES NOT address the protection of the computer's operations that stores the private FHE key. In other words, FHE defers the malware problem to the user's local computer; FHE does not address how to hinder malware on the local computer. Furthermore, some FHE keys are 1 Gigabyte and the plaintext-to-ciphertext expansion is 10,000 to 1 for just

6

100 bits of security [26, 27]. These memory requirements are not economically feasible for protecting instructions in hardware.

In [28], they present a general approach to addressing code-injection attacks in scripting and interpreted languages (e.g., web-based SQL injection), by randomizing the instructions. In [28], they do not address malware attacks at the machine instruction (physical hardware) level; there is no notion nor use of stability to build instruction opcodes and operands that are resistant to small changes.

Secure multi-party computation (MPC) [29, 30] enables a group to jointly perform a computation without disclosing any participant's private inputs. MPC does not address when Alice does not trust her own machine, or the machine instructions executed on her machine. Overall, none of these prior art methods address the malware problem, described in section 4.

7 MOTIVATING STABLE COMPUTATION

In the prior art, register machines execute one instruction at a time [17, 16]. Even if there is a procedure to assure that the register machine program is executing correctly, this friendly procedure may never execute due to just one rogue branch instruction. Typical programming languages (e.g., C, Fortran, Java, Lisp and Python) are Turing complete and depend upon branching instructions. While conditional branching instructions are not required for universal computation, Rojas's methods still use unconditional branching and program self-modification. Moreover, about 75% to 80% of the control flow instructions, executed on register machines, are conditional branch instructions. (See figure A.14 in [17].)

These observations suggest that a computer program's purpose can be subverted because the register machine behavior is not always invariant when small changes are made to one or more instructions. In contrast, the nervous systems of some organisms do not seem to share this weakness [32, 33].

Overall, the purpose of our invention(s) of stable computation are based on the following design principle: if a small or moderate change is made to a register machine program, then the program's purpose is stable; if a large change is made, the program can no longer execute. Our principle is partly based on the observation that it is generally far more difficult to detect if a small change has altered the purpose of a program. With a small change, the tampered register machine program still can execute, but does not perform the task that the original program was designed to accomplish. (Our unstable C program provides an example of a small 1-bit change.) For this reason, our invention(s) create stable computation that is also incomprehensible to malware authors so that it is far more challenging for malware to subvert the program without completely destroying its functionality.

8 AN UNSTABLE C PROGRAM

We demonstrate unstable computation with C source code that adds 3 integers. This C code shows how small changes to the address of only one instruction can substantially alter the program's behavior.

```
#include <stdio.h>
#define NUM_BITS 16
int pow2[NUM_BITS] = {0x8000, 0x4000, 0x2000, 0x1000, 0x800, 0x400, 0x200, 0x100,
                      0x80, 0x40, 0x20, 0x10, 0x8, 0x4, 0x2, 0x1};
```

```

int addition(int a, int b)
{
    return (a + b);
}
int multiply(int a, int b)
{
    return (a * b);
}
int exec_op(int* num, int n, int (*op) (int, int))
{
    int i, v = num[0];
    for(i = 1; i < n; i++)
    {
        v = op(v, num[i]);
    }
    return v;
}
void print_numbers(int* v, int n)
{
    int k;
    printf ("\n");
    for(k = 0; k < n; k++)
    {
        printf("%d ", v[k] );
    }
}
void print_binary(unsigned int v)
{
    int k;
    for(k = 0; k < NUM_BITS; k++)
    {
        if (v / pow2[k]) printf("1 ");
        else printf("0 ");
        v %= pow2[k];
    }
    printf("\n");
}
int bop(int* m, int n, char* f, int (*op) (int, int))
{
    int v = exec_op(m, n, op);
    printf ("\nresult = %d. address of %s = ", v, f);
    print_binary((unsigned int) op);
    return 0;
}
int main(int argc, char* argv[ ])
{
    int num [3] = {2, 3, 5};
    print_numbers (num, 3);
    printf("\n");
    bop(num, 3, "addition", addition);
    bop(num, 3, "multiply", multiply);
    return 0;
}

```

FIG. 14 shows an execution of the compiled C program: ADD. The sum of the numbers 2+3+5 is converted to the product 2*3*5, by flipping only one bit of the address of instruction addition. This C program exhibits unstable computation because a small change (flipping one bit) in the C program causes a substantial change to the outcome of its computation: namely: a sum equal to 10 is changed to a product equal to 30.

9 DESCRIPTION of FIGURES

In the following figures, although they may depict various examples of the invention, the invention is not limited to the examples depicted in the figures.

FIG. 1A shows an embodiment of an information system for sending and receiving hidden information or data. In some embodiments, the hidden information represents hidden machine instructions.

FIG. 1B shows an embodiment of a process for hiding information that can be used in the embodiment of FIG. 1A.

FIG. 1C shows an embodiment of an information system for sending and receiving hidden public keys.

FIG. 1D shows an embodiment of a process for hiding public keys that can be used in the embodiment of FIG. 1C.

FIG. 1E shows an embodiment of storing machine **180** that executes hiding process **184** to hide data **182** and store hidden data **186** in memory system **188**.

FIG. 1F shows an embodiment of hiding machine **190** that executes hiding process **194** to hide machine instruction **192** and store hidden machine instruction **196** in memory system **198**.

FIG. 2A shows an embodiment of a computer network transmitting hidden data or keys, hidden encrypted data or hidden metadata. In some embodiments, the transmission may be over the Internet or a part of a network that supports an infrastructure such as the electrical grid, a financial exchange, or a power plant, which can be used with the embodiment of FIG. 1A.

FIG. 2B shows an embodiment for hiding information, which includes a processor, memory and input/output system, that may be sending and/or receiving machines of FIG. 1A.

FIG. 3A shows an embodiment of a USB drive that can act as a sending machine and receiving machine to store and protect a user's data.

FIG. 3B shows an embodiment of an authentication token, which may include the sending and/or receiving machines of FIG. 1A, that contains a computer processor that can hide data or hide keys.

FIG. 4 shows a mobile phone embodiment 400 that hides wireless voice metadata and extracts wireless voice data that was hidden, which may include the sending and/or receiving machines of FIG. 1A. The mobile phone 500 is an embodiment that sends wireless hidden metadata, hidden encrypted data, or hidden keys to an automobile, which may include the sending and/or receiving machines of FIG. 1A.

FIG. 5 shows key(s) hidden in random noise where the probability distribution of the key is the same as the probability distribution of the noise.

FIG. 6A shows data hidden in random noise where the probability distribution of the data is the same as the probability distribution of the noise.

FIG. 6B shows a machine instruction hidden in random noise where the probability distribution of the machine instruction is the same as the probability distribution of the noise.

FIG. 7 shows a key hidden in random noise where the probability distribution of the key and the noise are not the same. The probability distribution of the key is somewhat biased.

FIG. 8 shows data hidden in random noise where the probability distribution of the data and the noise are not the same. The probability distribution of the data is more biased.

FIG. 9A shows an embodiment of a non-deterministic generator, based on quantum randomness. Nondeterministic generator 942 is based on the behavior of photons to help generate noise and in some embodiments one or more keys. Non-deterministic generator 942 contains a light emitting diode 946 that emits photons and a phototransistor 944 that absorbs photons.

FIG. 9B shows an embodiment of a non-deterministic generator, based on quantum randomness. Nondeterministic generator 952 is based on the behavior of photons to help generate noise and in some embodiments one or more keys. Non-deterministic generator 952 contains a light emitting diode 956 that emits photons and a photodiode 954 that absorbs photons.

FIG. 9C shows an embodiment of a deterministic generator 962, implemented with a machine. Deterministic generator 962 may generate one or more keys 970 or noise 972. Deterministic generator 962 has generator update instructions 966, one-way hash instructions 964 and one-way hash instructions 968.

FIG. 9D shows an embodiment of a hiding locator machine 980 with dynamic hiding locations 982, one-way hash Φ instructions 988, hiding locator instructions 986, one-way hash Ψ instructions 988, and noise 990.

FIG. 10 shows a light emitting diode, which emits photons and in some embodiments is part of the random number generator. The light emitting diode contains a cathode, a diode, an anode, one terminal pin connected to the cathode and one terminal pin connected to the anode, a p-layer of semiconductor, an active region, an n-layer of semiconductor, a substrate and a transparent plastic case.

FIG. 11 shows a scatter map that hides 128 bits of data inside of 128 bits of noise. In an embodiment, the data is a 128 bit public key.

FIG. 12 shows a data transformation that transforms 18 bits of data to a larger sequence of data.

FIG. 13 shows probabilities after Eve observes a hidden key or hidden data inside random noise. The hidden key or hidden noise is represented as S.

FIG. 14 shows an execution of the compiled C program: ADD, where the adversary Eve only has to flip one bit of one machine instruction to sabotage the program so that it multiplies instead of adds.

FIG. 15 shows Protected Machine Hardware (blue region) such that the adversary Eve cannot access and sabotage the instructions in this region of the computer hardware.

FIG. 16 shows the ratio of $n!$ (factorial) to 2^n for register sizes of $n=64$ bits, $n=128$ bits, and $n=256$ bits. The big numbers following the symbol '=' represent the calculation of

$$\frac{n!}{2^n}$$

for $n=04$, $n=128$, and $n=256$.

FIG. 17, titled Instruction Order, shows a table representing how a block of machine instructions S_{j_1}, S_{j_2}, \dots have been hidden in dummy instructions D_{r_1}, D_{r_2}, \dots after Procedure 7 is executed.

FIG. 18, titled 16 Bit Butterfly Network, shows a Butterfly network in hardware implementing a permutation of bits.

10 DETAILED DESCRIPTION

10.1 Information System

In this specification, the term "data" is broad and refers to any kind of information. In some embodiments, data may refer to plaintext information. In some embodiments, data may refer to voice information, transmitted with a landline phone or mobile phone. In some embodiments, data may refer to metadata. In some embodiments, data may refer to email or other information available on the Internet. In some embodiments, data may refer to the information in a sequence of values. In some embodiments, data may refer to the information in a sequence of bit values. In some embodiments, data may refer to the information in a sequence of numbers. In some embodiments, data may refer to the information in a sequence or collection of physical values or physical measurements. In some embodiments, data may refer to the information in a physical location (e.g., GPS coordinates of an auto or a mailing address in Venezia, Italia) or to the information in an abstract location—for example, a computer memory address or a virtual address. In some embodiments, data may refer to the information contained in Shakespeare's King Lear or Dostoevsky's Grand Inquisitor or Euclid's Elements. In some embodiments, data may refer to the information in Kepler's astronomical measurements or a collection of geophysical measurements. In some embodiments, data may refer to the information in a sequence of times or collection of times. In some embodiments, data may refer to the information in statistical data such as economic or insurance information. In some embodiments, data may refer to medical information (e.g., an incurable cancer diagnosis) or genetic information (e.g., that a person has the amino acid substitution causing sickle cell anemia). In some embodiments, data may refer to the

11

information in a photograph of friends or family or satellite photos. In some embodiments, data may refer to the information in a code or sequence of codes. In some embodiments, data may refer to the information in a sequence of language symbols for a language that has not yet been discovered or designed. In some embodiments, data may refer to financial information—for example, data may refer to a bid quote on a financial security, or an ask quote on a financial security. In some embodiments, data may refer to information about a machine or a collection of machines—for example, an electrical grid or a power plant. In some embodiments, data may refer to what electrical engineers sometimes call signal in information theory. In some embodiments, data may refer to a cryptographic key. In some embodiments, data may refer to a sequence or collection of computer program instructions (e.g., native machine instructions or source code information). In some embodiments, data may refer to a prime number or a mathematical formula or a mathematical invariant information. In some embodiments, data may refer to an internet protocol address or internet traffic information. In some embodiments, data may refer to a combination or amalgamation or synthesis of one or more of these types of aforementioned information.

In this specification, the term “noise” is information that is distinct from data and has a different purpose. Noise is information that helps hide the data so that the noise hinders the adversary Eve from finding or obtaining the data. This hiding of the data helps maintain the privacy of the data. In some embodiments, hiding the data means rearranging or permuting the data inside the noise. An example of data is a key. Hiding a key inside noise helps protect the privacy of the key; the key may subsequently help execute a cryptographic algorithm by a first party (e.g., Alice) or a second party (e.g., Bob).

In this specification, the term “location” may refer to geographic locations and/or storage locations. A particular storage location may be a collection of contiguous and/or noncontiguous locations on one or more machine readable media. Two different storage locations may refer to two different sets of locations on one or more machine-readable media in which the locations of one set may be intermingled with the locations of the other set.

In this specification, the term “machine-readable medium” refers to any non-transitory medium capable of carrying or conveying information that is readable by a machine. One example of a machine-readable medium is a computer-readable medium. Another example of a machine-readable medium is paper having holes that are detected that trigger different mechanical, electrical, and/or logic responses. The term machine-readable medium also includes media that carry information while the information is in transit from one location to another, such as copper wire and/or optical fiber and/or the atmosphere and/or outer space.

In this specification, the term “process” refers to a series of one or more operations. In an embodiment, “process” may also include operations or effects that are best described as non-deterministic. In an embodiment, “process” may include some operations that can be executed by a digital computer program and some physical effects that are non-deterministic, which cannot be executed by a digital computer program and cannot be performed by a finite sequence of processor instructions.

In this specification, the machine-implemented processes implement algorithms and non-deterministic processes on a machine. The formal notion of “algorithm” was introduced in Turing’s work and refers to a finite machine that executes

12

a finite number of instructions with finite memory. In other words, an algorithm can be executed with a finite number of machine instructions on a processor. “Algorithm” is a deterministic process in the following sense: if the finite machine is completely known and the input to the machine is known, then the future behavior of the machine can be determined. However, there is quantum random number generator (QRNG) hardware (FIG. 9A, FIG. 9B, and FIG. 10) in embodiments that measure quantum effects from photons (or other physically non-deterministic processes), whose physical process is nondeterministic. The recognition of non-determinism produced by quantum randomness and other quantum embodiments is based on many years of experimental evidence and statistical testing. Furthermore, the quantum theory-derived from the Kochen-Specker theorem and its extensions [37, 38]-predicts that the outcome of a quantum measurement cannot be known in advance and cannot be generated by a Turing machine (digital computer program). As a consequence, a physically non-deterministic process cannot be generated by an algorithm: namely, a sequence of operations executed by a digital computer program. FIG. 9A, FIG. 9B, and FIG. 10 show an embodiment of a non-deterministic process arising from quantum events; that is, the emission and absorption of photons.

Some examples of physically non-deterministic processes are as follows. In some embodiments that utilize non-determinism, photons strike a semitransparent mirror and can take two or more paths in space. In one embodiment, if the photon is reflected by the semitransparent mirror, then it takes on one bit value $b \in \{0, 1\}$; if the photon passes through by the semitransparent mirror, then the non-deterministic process produces another bit value 1-6. In another embodiment, the spin of an electron may be sampled to generate the next non-deterministic bit. In still another embodiment, a protein, composed of amino acids, spanning a cell membrane or artificial membrane, that has two or more conformations can be used to detect non-determinism: the protein conformation sampled may be used to generate a non-deterministic value in $\{0, \dots, n-1\}$ where the protein has n distinct conformations. In an alternative embodiment, one or more rhodopsin proteins could be used to detect the arrival times of photons and the differences of arrival times could generate non-deterministic bits. In some embodiments, a Geiger counter may be used to sample non-determinism.

In this specification, the term “photodetector” refers to any type of device or physical object that detects or absorbs photons. A photodiode is an embodiment of a photodetector. A phototransistor is an embodiment of a photodetector. A rhodopsin protein is an embodiment of a photodetector.

In this specification, the term “key” is a type of information and is a value or collection of values to which one or more operations are performed. In some embodiments, one or more of these operations are cryptographic operations. $\{0, 1\}^n$ is the set of all bit-strings of length n . When a key is represented with bits, mathematically a n -bit key is an element of the collection $\{0, 1\}^n$ which is the collection of strings of 0’s and 1’s of length n . For example, the string of 0’s and 1’s that starts after this colon is a 128-bit key:
 01100001 11000110 01010011 01110001 11000101
 10001110 11011001 11010101 01011001 01100100
 10110010 10101010 01101101 10000111 10101011
 00010111. In an embodiment, $n=3000$ so that a key is a string of 3000 bits.

In other embodiments, a key may be a sequence of values that are not represented as bits. Consider the set $\{A, B, C, D, E\}$. For example, the string that starts after this colon is

13

a 40-symbol key selected from the set {A, B, C, D, E}: ACDEB AADBC EAEBB AAECB ADDCB BDCCE ACECB EACAE. In an embodiment, a key could be a string of length n selected from {A, B, C, D, E} ^{n} . In an embodiment, $n=700$ so that the key is a string of 700 symbols where each symbol is selected from {A, B, C, D, E}. In some embodiments, a key is a collection of one or more values, that specifies how a particular encryption function will encrypt a message. For example, a key may be a sequence of 0's and 1's that are bitwise exclusive-or'ed with the bits that comprise a message to form the encrypted message.

In some embodiments, hidden data (key) **109** in FIG. 1A may be read as input by processor system **258**, that executes instructions which perform a cryptographic algorithm. In some embodiments, hidden data (key) **132** in FIG. 1B, may be read as input by processor system **258**, that executes instructions which perform a cryptographic algorithm. Symmetric cryptography typically is implemented with a block cipher or a stream cipher. In another embodiment, a key K may be a sequence of values that a stream cipher reads as input so that Alice can encrypt a message M as $\mathcal{E}(K, M)$ with this key and Bob can decrypt $\mathcal{E}(K, M)$ message. In the expression $\mathcal{E}(K, M)$, K represents the key, M represents the message and \mathcal{E} represents the encryption method.

In another embodiment, a key may be a sequence of values that a block cipher reads as input in order to encrypt a message with the block cipher encryption algorithm \mathcal{E} . In another embodiment, a key may be a sequence of values that a block cipher reads as input in order to decrypt an encrypted message with the block cipher's decryption algorithm \mathcal{D} . If Eve does not know that key, then it is difficult for Eve to decrypt the encrypted message $\mathcal{E}(K, M)$. AES is a common block cipher algorithm that reads 256-bit keys as input. Serpent is also a block cipher algorithm that reads 256-bit

keys as input. In other embodiments, the key be a public key. In some embodiments, a key may refer to a public key for the RSA public-key algorithm [41]. In this case, a key is a huge prime number. In some embodiments, random generator **128** generates a key that is subsequently hidden by scatter map instructions **130**.

FIG. 1A shows an information system **100** for hiding information in a manner that is expected to be secure. In this specification, data will sometimes refer to information that has not yet been hidden or encrypted. Information system **100** includes data **104** (not hidden information), and hiding process **106**, a sending machine **102**, hidden data (hidden information) **109** and a transmission path **110**, a receiving machine **112**, extraction process **116**, extracted data **114**. In other embodiments, information system **100** may not have all of the components listed above or may have other components instead of and/or in addition to those listed above.

Information system **100** may be a system for transmitting hidden data. Data **104** refers to information that has a purpose and that has not been hidden yet. In some embodiments, data is intended to be delivered to another location, software unit, machine, person, or other entity.

In some embodiments, data **104** is voice metadata that has not yet been hidden. Voice metadata may contain the IP address of the sending (calling) phone and also the IP address of the receiving phone. Voice metadata may contain the time of the call and the date. Some embodiments of a mobile phone are shown in FIG. 4. In other embodiments, data **104** is email metadata or text metadata or browser metadata.

14

In an embodiment, data may be unhidden information being transmitted wirelessly between satellites. Data may be represented in analog form in some embodiments and may be represented in digital form. In an embodiment, the sound waves transmitted from a speaker's mouth into a mobile phone microphone are data. The representation of this data information before reaching the microphone is in analog form. Subsequently, the data information may be digitally sampled so it is represented digitally after being received by the mobile phone microphone. In general, data herein refers to any kind of information that has not been hidden or encrypted and that has a purpose.

In information system **100**, noise helps hide the data. It may be desirable to keep the contents of data **104** private or secret. Consequently, it may be desirable to hide data **104**, so that the transmitted information is expected to be unintelligible to an unintended recipient should the unintended recipient attempt to read and/or extract the hidden data transmitted. Data **104** may be a collection of multiple, not yet hidden information blocks, an entire message of data, a segment of data (information), or any other portion of a data.

Hiding process **106** may be a series of steps that are performed on data **104**. In one embodiment, the term "process" refers to one or more instructions for sending machine **102** to execute the series of operations that may be stored on a machine-readable medium. Alternatively, the process may be carried out by and therefore refer to hardware (e.g., logic circuits) or may be a combination of instructions stored on a machine-readable medium and hardware that cause the operations to be executed by sending machine **102** or receiving machine **112**. Data **104** may be input for hiding process **106**. The steps that are included in hiding process **106** may include one or more mathematical operations and/or one or more other operations.

As a post-processing step, one-way hash function **948** may be applied to a sequence of random events such as quantum events (non-deterministic) generated by non-deterministic generator **942** in FIG. 9A. As a post-processing step, one-way hash function **948** may be applied to a sequence of random events such as quantum events (non-deterministic) generated by non-deterministic generator **952** in FIG. 9B.

In FIG. 1B hiding process **122** may implement hiding process **106** in FIG. 1A. In some embodiments, cipher instructions **129** may first encrypt the data **124** and subsequently scatter map instructions **130** hide the encrypted data to produce hidden encrypted data **132** before sending machine **102** transmits the hidden data via transmission path **110**. In some embodiments, data transformation instructions **126** may transform data **124** before scatter map process instructions **130** are applied to this transformed data. In some embodiments, scatter map process instructions **130** are at least part of the hiding process. In some embodiments, data **124** is transformed by data transformation instructions **126** and encrypted by cipher instructions **129** before scatter map process instructions **130** are applied to this transformed and encrypted data.

In some embodiments, as shown in FIG. 1B, random generator **128** is used to help generate the scatter map that helps perform scatter map process instructions **130**. In some embodiments, random generator **128** generates noise that is used by scatter map process instructions **130** to hide data **124** that has previously been transformed by data transformation instructions **126** and/or encrypted by cipher instructions **129**. In some embodiments, random generator **128** generates one or more keys as input to cipher instructions **129** that are applied to data **124**. In some embodiments, random genera-

15

tor 128 generates one or more keys that are hidden in random noise, generated by random generator 128.

In some embodiments, hiding process 106 requests random generator 128 to help generate one or more keys (shown in cipher instructions 129) for encrypting at least part of data 104. In an embodiment, nondeterministic generator 942 (FIG. 9A) may be part of random generator 128. In an embodiment, nondeterministic generator 952 (FIG. 9B) may be part of random generator 128.

Sending machine 102 may be an information machine that handles information at or is associated with a first location, software unit, machine, person, sender, or other entity. Sending machine 102 may be a computer, a phone, a mobile phone, a telegraph, a satellite, or another type of electronic device, a mechanical device, or other kind of machine that sends information. Sending machine 102 may include one or more processors and/or may include specialized circuitry for handling information. Sending machine 102 may receive data 104 from another source (e.g., a transducer such as a microphone which is inside mobile phone 402 or 502 of FIG. 4), may produce all or part of data 104, may implement hiding process 106, and/or may transmit the output to another entity. In another embodiment, sending machine 102 receives data 104 from another source, while hiding process 106 and the delivery of the output of hiding process 106 are implemented manually. In another embodiment, sending machine 102 implements hiding process 106, having data 104 entered, via a keyboard (for example) or via a mobile phone microphone, into sending machine 102. In another embodiment, sending machine 102 receives output from hiding process 106 and sends the output to another entity.

Sending machine 102 may implement any of the hiding processes described in this specification. Hiding process 106 may include any of the hiding processes described in this specification. For example, hiding process 106 may implement any of the embodiments of the hiding processes 1 or 3, as described in section 10.7; hiding process 106 may implement any of the embodiments of the hiding process 5, as described in section 10.12; hiding process 106 may implement any of the embodiments of the hiding processes 6 or 7, as described in section 10.13; hiding process 106 may implement any of the embodiments of the hiding processes 8 or 9, as described in section 10.15. In some embodiments, hidden data 132, shown in FIG. 1B, includes at least some data 124 that was hidden by the scatter map process instructions 130 that is a part of hiding process 122.

Transmission path 110 is the path taken by hidden data 109 to reach the destination to which hidden data 109 was sent. Transmission path 110 may include one or more networks, as shown in FIG. 2A. In FIG. 2A, network 212 may help support transmission path 110. For example, transmission path 110 may be the Internet, which is implemented by network 212; for example, transmission path 110 may be wireless using voice over Internet protocol, which is implemented by network 212. Transmission path 110 may include any combination of any of a direct connection, hand delivery, vocal delivery, one or more Local Area Networks (LANs), one or more Wide Area Networks (WANs), one or more phone networks, including paths under the ground via fiber optics cables and/or one or more wireless networks, and/or wireless inside and/or outside the earth's atmosphere.

Receiving machine 112 may be an information machine that handles information at the destination of an hidden data 109. Receiving machine 112 may be a computer, a phone, a telegraph, a router, a satellite, or another type of electronic device, a mechanical device, or other kind of machine that receives information. Receiving machine 112 may include

16

one or more processors and/or specialized circuitry configured for handling information, such as hidden data 109. Receiving machine 112 may receive hidden data 109 from another source and/or reconstitute (e.g., extract) all or part of hidden data 109. Receiving machine 112 may implement any of the hiding processes described in this specification and is capable of extracting any message hidden by sending machine 102 and hiding process 106.

In one embodiment, receiving machine 112 only receives hidden data 109 from transmission path 110, while hiding process 106 is implemented manually and/or by another information machine. In another embodiment, receiving machine 112 implements extraction process 116 that reproduces all or part of data 104, referred to as extracted data 114 in FIG. 1A. In another embodiment, receiving machine 112 receives hidden data 109 from transmission path 110, and reconstitutes all or part of extracted data 114 using extraction process 116. Extraction process 116 may store any of the processes of hiding information described in this specification. Extraction process 116 may include any of the hiding processes described in this specification.

Receiving machine 112 may be identical to sending machine 102. For example, receiving machine 112 may receive data 104 from another source, produce all or part of data 104, and/or implement hiding process 106. Similar to sending machine 102, receiving machine 112 may create keys and random noise and random data. Receiving machine 112 may transmit the output of extraction process 116, via transmission path 110 to another entity and/or receive hidden data 109 (via transmission path 110) from another entity. Receiving machine 112 may present hidden data 109 for use as input to extraction process 116.

10.2 Public Key Information System

In this specification, the term "public key" refers to any kind of public key used in public key cryptography. In an embodiment, "public key" refers to an RSA public key. In an embodiment, "public key" refers to an elliptic curve public key. In an embodiment, "public key" refers to a lattice public key. In an embodiment, "public key" refers to a Goppa code public key.

In this specification, the term "public key" is a type of information and is a value or collection of values to which one or more operations are performed. In some embodiments, one or more of these operations are cryptographic operations. $\{0, 1\}^n$ is the set of all bit-strings of length n . When a public key is represented with bits, mathematically a n -bit key is an element of the collection $\{0, 1\}^n$ which is the collection of strings of 0's and 1's of length n . For example, the string of 0's and 1's that starts after this colon is a 128-bit key: 01100001 11000110 01010011 01110001 11000101 10001110 11011001 11010101 01011001 01100100 10110010 10101010 01101101 10000111 10101011 00010111. In an embodiment, $n=3000$ so that a key is a string of 3000 bits.

In other embodiments, a public key may be a sequence of values that are not represented as bits. Consider the set $\{A, B, C, D, E\}$. For example, the string that starts after this colon is a 40-symbol key selected from the set $\{A, B, C, D, E\}$: ACDEB AADBC EAEBB AAECB ADDCB BDCCE ACECB EACAE.

In an embodiment, a key could be a string of length n selected from $\{A, B, C, D, E\}^n$. In an embodiment, $n=700$ so that the key is a string of 700 symbols where each symbol is selected from $\{A, B, C, D, E\}$. In some embodiments, hidden public key(s) 149 in FIG. 1C may be read as input by

processor system **258** in FIG. **2B**, that executes instructions which perform extraction process **156** in FIG. **1C**. In some embodiments, hidden public key(s) **172** in FIG. **1D**, may be read as input by processor system **258**, that executes instructions which perform extraction process **156**.

In some embodiments, public key(s) **144** are RSA public key(s), which is a well-known public key cryptography [41]. RSA is described from the perspective of Alice. Alice chooses two huge primes p_A and q_A . Alice computes $n_A = p_A q_A$ and a random number r_A which has no common factor with $(p_A - 1)(q_A - 1)$. In other words, 1 is the greatest common divisor of r_A and $(p_A - 1)(q_A - 1)$. The Euler-phi function is defined as follows. If $k=1$, then $\phi(k)=1$; if $k>1$, then $\phi(k)$ is the number positive integers i such that $i \leq k$ and i and k are relatively prime. Relatively prime means the greatest common divisor of i and k is 1. The positive integer e_A is randomly selected such that e_A is relatively prime to $\phi(n_A)$.

Alice computes $\phi(N_A) = N_A + 1 - p_A - q_A$. Alice computes the multiplicative inverse of r_A modulo $\phi(n_A)$; the multiplicative inverse is $d_A = e_A^{-1}$ modulo $\phi(n_A)$. Alice makes public her public key (n_A, r_A) : that is, the two positive integers (n_A, r_A) are Alice's public key.

In an embodiment, random generator **168** generates $r_1 \dots r_p$ which is input to private key instructions **164**. In an embodiment that hides RSA public keys, private key instruction **164** use $r_1 \dots r_p$ to find two huge primes p_A and q_A and a random number r_A relatively prime to $(p_A - 1)(q_A - 1)$.

In an embodiment, random generator **168** and private key instructions **164** generate two huge primes p_A and q_A ; compute $n_A = p_A q_A$; and randomly choose e_A that is relatively prime to $\phi(n_A)$. In an embodiment, private key instructions **164** compute $d_A = e_A^{-1}$ modulo $\phi(N_A)$. In an embodiment, an RSA private key is (n_A, d_A) . In an embodiment that hides RSA public keys, public key instructions **166** compute RSA public key (n_A, r_A) . In an embodiment, positive integer n_A is a string of 4096 bits and r_A is a string of 4096 bits.

FIG. **1C** shows an information system **140** for hiding public keys in a manner that is expected to be secure. In this specification, open public key will sometimes refer to a public key that has not yet been hidden and extracted public key will refer to a public key that was previously hidden and extracted from the noise. Information system **140** includes one or more private keys **103** and one or more corresponding public keys **144**, and hiding process **146**, a sending machine **142**, hidden key(s) **149** and a transmission path **150**, a receiving machine **152**, extraction process **156**, extracted public key(s) **154**. In other embodiments, information system **140** may not have all of the components listed above or may have other components instead of and/or in addition to those listed above.

Information system **140** may be a system for transmitting hidden public key(s). Public key(s) **144** refers to information that has a purpose and that has not been hidden yet. In some embodiments, public key(s) **144** is intended to be delivered to another location, software unit, machine, person, or other entity.

In some embodiments, public key(s) **144** may serve as part of a key exchange that has not yet been hidden. In an embodiment, public key(s) **144** may be unhidden information before it is hidden and transmitted wirelessly between satellites. Public key(s) **144** may be represented in analog form in some embodiments and may be represented in digital form. In an embodiment, the public key(s) may be one or more RSA public keys based on huge prime numbers.

In an another embodiment, the public key(s) may be one or more elliptic curve public keys, computed from an elliptic curve over a finite field.

In information system **140**, noise helps hide public key(s) **144**. Although they are public, it may be desirable to keep public key(s) **144** private or secret from Eve. For example, it is known that Shor's quantum computing algorithm can compute in polynomial time the corresponding private key of a RSA public key. As another example, an analogue of Shor's algorithm can compute in polynomial time the corresponding private key of an elliptic curve public key. If Eve has a quantum computer that computes enough qubits, then Eve could find the private key of an RSA public key that is disclosed to Eve and consequently breach the security of information system **140**. One or more RSA public keys could be hidden in noise to protect them from Eve's quantum computer. Consequently, it may be desirable to hide public key(s) **144**, so that the transmitted information is expected to be unintelligible to an unintended recipient should the unintended recipient attempt to read and/or extract the hidden public key(s) **149** transmitted. Public key(s) **144** may be a collection of multiple, not yet hidden blocks of information, an entire sequence of public keys, a segment of public keys, or any other portion of one or more public keys. When there is more than one public key, public keys **144** may be computed from distinct commutative groups, as described in section 10.16. For example, one commutative group may be based on an elliptic curve over a finite field; another commutative group may be based on multiplication modulo, as used in RSA.

Hiding process **146** may be a series of steps that are performed on public keys **144**. In one embodiment, the term "process" refers to one or more instructions for sending machine **142** to execute the series of operations that may be stored on a machine-readable medium. Alternatively, the process may be carried out by and therefore refer to hardware (e.g., logic circuits) or may be a combination of instructions stored on a machine-readable medium and hardware that cause the operations to be executed by sending machine **142** or receiving machine **152**. Public key(s) **144** may be input for hiding process **146**. The steps that are included in hiding process **146** may include one or more mathematical operations and/or one or more other operations.

As a post-processing step, one-way hash function **948** may be applied to a sequence of random events such as quantum events (non-deterministic) generated by non-deterministic generator **942** in FIG. **9A**. As a post-processing step, one-way hash function **948** may be applied to a sequence of random events such as quantum events (non-deterministic) generated by non-deterministic generator **952** in FIG. **9B**.

In FIG. **1D** hiding process **162** may implement hiding process **146** in FIG. **1C**. In some embodiments, random generator **168** help generate noise that is used by scatter map instructions **170**. In some embodiments, hiding process **162** requests random generator **168** and private key instructions **164** to help generate one or more private keys **103** that are used to compute public keys **144**. In an embodiment, non-deterministic generator **942** (FIG. **9A**) may be part of random generator **168**. In an embodiment, non-deterministic generator **952** (FIG. **9B**) may be part of random generator **168**.

Sending machine **142** may be an information machine that handles information at or is associated with a first location, software unit, machine, person, sender, or other entity. Sending machine **142** may be a computer, a phone, a mobile

19

phone, a telegraph, a satellite, or another type of electronic device, a mechanical device, or other kind of machine that sends information. Sending machine 142 may include one or more processors and/or may include specialized circuitry for handling information. Sending machine 142 may receive public key(s) 144 from another source (e.g., a transducer such as a microphone which is inside mobile phone 402 or 502 of FIG. 4), may produce all or part of public key(s) 144, may implement hiding process 146, and/or may transmit the output to another entity. In another embodiment, sending machine 142 receives public key(s) 144 from another source, while hiding process 146 and the delivery of the output of hiding process 146 are implemented manually. In another embodiment, sending machine 142 implements hiding process 146, having public key(s) 144 entered, via a keyboard (for example) or via a mobile phone microphone, into sending machine 142. In another embodiment, sending machine 142 receives output from hiding process 146 and sends the output to another entity.

Sending machine 142 may implement any of the hiding processes described in this specification. Hiding process 146 may include any of the hiding processes described in this specification. For example, hiding process 146 may implement any of the embodiments of the hiding processes 3 in section 10.7 and processes 10, 11 in section 10.17.

In some embodiments, hiding process 162, shown in FIG. 1D, generates one or more private keys p_1, \dots, p_m from private key instructions 164 and random generator 168; computes one or more public keys k_1, \dots, k_m with public key instructions 166; and scatter map instructions 170 hide one or public keys in noise $r_1 \dots r_p$ generated from random generator 168.

Transmission path 150 is the path taken by hidden public key(s) 149 to reach the destination to which hidden public key(s) 149 was sent. Transmission path 150 may include one or more networks, as shown in FIG. 2A. In FIG. 2A, network 212 may help support transmission path 150. For example, transmission path 150 may be the Internet, which is implemented by network 212; for example, transmission path 150 may be wireless using voice over Internet protocol, which is implemented by network 212. Transmission path 150 may include any combination of any of a direct connection, hand delivery, vocal delivery, one or more Local Area Networks (LANs), one or more Wide Area Networks (WANs), one or more phone networks, including paths under the ground via fiber optics cables and/or one or more wireless networks, and/or wireless inside and/or outside the earth's atmosphere.

Receiving machine 152 may be an information machine that handles information at the destination of an hidden public key(s) 149. Receiving machine 152 may be a computer, a phone, a telegraph, a router, a satellite, or another type of electronic device, a mechanical device, or other kind of machine that receives information. Receiving machine 152 may include one or more processors and/or specialized circuitry configured for handling information, such as hidden public key(s) 149. Receiving machine 152 may receive hidden public key(s) 149 from another source and/or reconstitute (e.g., extract) all or part of hidden public key(s) 149. Receiving machine 152 may implement any of the hiding processes described in this specification and is capable of extracting any message hidden by sending machine 142 and hiding process 146.

In one embodiment, receiving machine 152 only receives hidden public key 149 from transmission path 150, while hiding process 146 is implemented manually and/or by another information machine. In another embodiment, receiving machine 152 implements extraction process 156

20

that reproduces all or part of public key(s) 144, referred to as extracted public key(s) 154 in FIG. 1C. In another embodiment, receiving machine 152 receives hidden public key(s) 149 from transmission path 150, and reconstitutes all or part of extracted public key(s) 154 using extraction process 156. Extraction process 156 may store any of the processes of hiding information described in this specification. Extraction process 156 may include any of the hiding processes described in this specification.

Receiving machine 152 may be identical to sending machine 142. For example, receiving machine 152 may receive 144 from another source, produce all or part of public key(s) 144, and/or implement hiding process 146. Similar to sending machine 142, receiving machine 152 may create keys and random noise and random public key(s). Receiving machine 152 may transmit the output of extraction process 156, via transmission path 150 to another entity and/or receive hidden public key(s) 149 (via transmission path 150) from another entity. Receiving machine 152 may present hidden public key(s) 149 for use as input to extraction process 156.

10.3 PROCESSOR, MEMORY and INPUT/OUTPUT HARDWARE

Information system 200 illustrates some of the variations of the manners of implementing information system 100. Sending machine 202 is one embodiment of sending machine 101. Sending machine 202 may be a secure USB memory storage device as shown in FIG. 3A. Sending machine 202 may be an authentication token as shown in FIG. 3B. A mobile phone embodiment of sending machine 202 is shown in FIG. 4.

Sending machine 202 or sending machine 400 may communicate wirelessly with computer 204. In an embodiment, computer 204 may be a call station for receiving hidden data 109 from sending machine 400. A user may use input system 254 and output system 252 of sending machine (mobile phone) 400 to transmit hidden voice data or hidden metadata to a receiving machine that is a mobile phone. In an embodiment, input system 254 in FIG. 2B includes a microphone that is integrated with sending machine (mobile phone) 400. In an embodiment, output system 252 in FIG. 2B includes a speaker that is integrated with sending machine (mobile phone) 400. In another embodiment, sending machine 202 is capable of being plugged into and communicating with computer 204 or with other systems via computer 204.

Computer 204 is connected to system 210, and is connected, via network 212, to system 214, system 216, and system 218, which is connected to system 220. Network 212 may be any one or any combination of one or more Local Area Networks (LANs), Wide Area Networks (WANs), wireless networks, telephones networks, and/or other networks. System 218 may be directly connected to system 220 or connected via a LAN to system 220. Network 212 and system 214, 216, 218, and 220 may represent Internet servers or nodes that route hidden data (e.g., hidden voice data or hidden metadata) received from sending machine 400 shown in FIG. 4. In FIG. 2A, system 214, 216, 218, and system 220 and network 212 may together serve as a transmission path 110 for hidden data 109. In an embodiment, system 214, 216, 218, and system 220 and network 212 may execute the Internet protocol stack in order to serve as transmission path 110 for hidden data 109. In an embodiment, hidden data 109 may be voice data. In an embodiment, hidden data 109 may be routing data. In an embodiment,

21

hidden data 109 may be π CP/IP data. In an embodiment, hidden data 109 may be metadata. In an embodiment, hidden data 109 may be email. In an embodiment, hidden data 109 may be text data sent from sending machine 400.

In FIG. 1B, hiding process 122 may be implemented by any of, a part of any of, or any combination of any of system 210, network 212, system 214, system 216, system 218, and/or system 220. As an example, routing information of transmission path 110 may be hidden with hiding process 122 that executes in system computer 210, network computers 212, system computer 214, system computer 216, system computer 218, and/or system computer 220. Hiding process 106 may be executed inside sending machine 400 and extraction process 116 may be executed inside receiving machine 400 in FIG. 4.

In an embodiment, hiding process 106 and extraction process 116 execute in a secure area of processor system 258 of FIG. 2B. In an embodiment, specialized hardware in processor system 258 may be implemented to speed up the computation of scatter map instructions 130 in FIG. 1B. In an embodiment, this specialized hardware in processor system 258 may be embodied as an ASIC (application specific integrated circuit) that computes SHA-1 and/or SHA-512 and/or Keccak and/or BLAKE and/or JH and/or Skein that help execute one-way hash function 948 in non-deterministic generator 942 or one-way hash function 958 in non-deterministic generator 952 or one-way hash instructions 964 in deterministic generator 962.

In an embodiment, specialized hardware in processor system 258 may be embodied as an ASIC (application specific integrated circuit) that computes SHA-1 and/or SHA-512 and/or Keccak and/or BLAKE and/or JH and/or Skein that help execute the HMAC function in process 6 named Hiding One or More Keys with Authentication or help execute process 7 named Hiding Encrypted Data Elements with Authentication. An ASIC chip can increase the execution speed and protect the privacy of hiding process 106 and extraction process 116.

In an embodiment, input system 254 of FIG. 2B receives voice data and sends the voice data to processor system 258 where the voice data or voice metadata is hidden. Output system 252 sends the hidden voice data 109 to a telecommunication network 212. In an embodiment, memory system 256 stores scatter map instructions 130, data transformation instructions 126, and cipher instructions 129.

In an embodiment, memory system 256 of FIG. 2B stores scatter map instructions 132. In an embodiment, memory system 256 stores hidden data or hidden metadata that is waiting to be sent to output system 252 and sent out along transmission path 110, routed and served by system computers 210, 214, 216, 218 and 220 and network 212.

10.4 One-Way Hash Functions

In FIG. 9A, 9B, 9C, or 9D, one-way hash function 948, 958, 964, or 984 may include one or more one-way functions. A one-way hash function Φ , has the property that given an output value z , it is computationally intractable to find an information element m_z such that $\Phi(m_z)=z$. In other words, a one-way function Φ is a function that can be easily computed, but that its inverse Φ^{-1} is computationally intractable to compute [?]. A computation that takes 10^{101} computational steps is considered to have computational intractability of 10^{101} .

More details are provided on computationally intractable. In an embodiment, there is an amount of time T that encrypted information must stay secret. If encrypted infor-

22

mation has no economic value or strategic value after time T , then computationally intractable means that the number of computational steps required by all the world's computing power will take more time to compute than time T . Let $C(t)$ denote all the world's computing power at the time t in years.

Consider an online bank transaction that encrypts the transaction details of that transaction. Then in most embodiments, the number of computational steps that can be computed by all the world's computers for the next 30 years is in many embodiments likely to be computationally intractable as that particular bank account is likely to no longer exist in 30 years or have a very different authentication interface.

To make the numbers more concrete, the 2013 Chinese supercomputer that broke the world's computational speed record computes about 33,000 trillion calculations per second [42]. If $T=1$ one year and we can assume that there are at most 1 billion of these supercomputers. (This can be inferred from economic considerations, based on a far too low 1 million dollar price for each supercomputer. Then these 1 billion supercomputers would cost 1,000 trillion dollars). Thus, $C(2014) \times 1$ year is less than $10^9 \times 33 \times 10^{15} \times 3600 \times 24 \times 365 = 1.04 \times 10^{33}$ computational steps.

As just discussed, in some embodiments and applications, computationally intractable may be measured in terms of how much the encrypted information is worth in economic value and what is the current cost of the computing power needed to decrypt that encrypted information. In other embodiments, economic computational intractability may be useless. For example, suppose a family wishes to keep their child's whereabouts unknown to violent kidnappers. Suppose $T=100$ years because it is about twice their expected lifetimes. Then $100 \text{ years} \times C(2064)$ is a better measure of computationally intractable for this application. In other words, for critical applications that are beyond an economic value, one should strive for a good estimate of the world's computing power.

One-way functions that exhibit completeness and a good avalanche effect or the strict avalanche criterion are preferable embodiments: these properties are favorable for one-way hash functions. The definition of completeness and a good avalanche effect are quoted directly from [43]:

If a cryptographic transformation is complete, then each ciphertext bit must depend on all of the plaintext bits. Thus, if it were possible to find the simplest Boolean expression for each ciphertext bit in terms of plaintext bits, each of those expressions would have to contain all of the plaintext bits if the function was complete. Alternatively, if there is at least one pair of n -bit plaintext vectors X and X_i that differ only in bit i , and $f(X)$ and $f(X_i)$ differ at least in bit j for all $\{(i, j): 1 \leq i, j \leq n\}$, the function f must be complete.

For a given transformation to exhibit the avalanche effect, an average of one half of the output bits should change whenever a single input bit is complemented. In order to determine whether a $m \times n$ (m input bits and n output bits) function f satisfies this requirement, the 2^m plaintext vectors must be divided into 2^{m-1} pairs, X and X_i such that X and X_i differ only in bit i . Then the 2^{m-1} exclusive-or sums $V_i = f(X) \oplus f(X_i)$ must be calculated. These exclusive-or sums will be referred to as avalanche vectors, each of which contains n bits, or avalanche variables.

If this procedure is repeated for all i such that $1 \leq i \leq m$ and one half of the avalanche variables are equal to 1 for each i , then the function f has a good avalanche effect. Of course this method can be pursued only if m is fairly small;

otherwise, the number of plaintext vectors becomes too large. If that is the case then the best that can be done is to take a random sample of plaintext vectors X , and for each value i calculate all avalanche vectors V_i . If approximately one half the resulting avalanche variables are equal to 1 for values of i , then we can conclude that the function has a good avalanche effect.

A hash function, also denoted as Φ , is a function that accepts as its input argument an arbitrarily long string of bits (or bytes) and produces a fixed-size output of information. The information in the output is typically called a message digest or digital fingerprint. In other words, a hash function maps a variable length m of input information to a fixed-sized output, $\Phi(m)$, which is the message digest or information digest. Typical output sizes range from 160 to 512 bits, but can also be larger. An ideal hash function is a function, whose output is uniformly distributed in the following way: Suppose the output size of Φ is n bits. If the message m is chosen randomly, then for each of the 2^n possible outputs z , the probability that $\Phi(m)=z$ is 2^{-n} . In an embodiment, the hash functions that are used are one-way.

A good one-way hash function is also collision resistant. A collision occurs when two distinct information elements are mapped by the one-way hash function Φ to the same digest. Collision resistant means it is computationally intractable for an adversary to find collisions: more precisely, it is computationally intractable to find two distinct information elements m_1, m_2 where $m_1 \neq m_2$ and such that $\Phi(m_1)=\Phi(m_2)$.

A number of one-way hash functions may be used to implement one-way hash function **148**. In an embodiment, SHA-512 can implement one-way hash function **148**, designed by the NSA and standardized by NIST [44]. The message digest size of SHA-512 is 512 bits. Other alternative hash functions are of the type that conform with the standard SHA-384, which produces a message digest size of 384 bits. SHA-1 has a message digest size of 160 bits. An embodiment of a one-way hash function **148** is Keccak [45]. An embodiment of a one-way hash function **148** is BLAKE [46]. An embodiment of a one-way hash function **148** is Grøstl [47]. An embodiment of a one-way hash function **148** is JH [48]. Another embodiment of a one-way hash function is Skein [49].

10.5 Non-Deterministic Generators

FIG. 9A shows an embodiment of a non-deterministic generator **942** arising from quantum events: that is, in some embodiments random generator **128** in FIG. 1B uses the emission and absorption of photons for its non-determinism. In FIG. 9A, phototransistor **944** absorbs photons emitted from light emitting diode **954**. In an embodiment, the photons are produced by a light emitting diode **946**. In FIG. 9B, non-deterministic generator **952** has a photodiode **954** that absorbs photons emitted from light emitting diode **956**. In an embodiment, the photons are produced by a light emitting diode **956**.

FIG. 10 shows a light emitting diode (LED) **1002**. In an embodiment, LED **1002** emits photons and is part of the non-deterministic generator **942** (FIG. 9A). In an embodiment, LED **1002** emits photons and is part of the non-deterministic generator **952** (FIG. 9B). LED **1002** contains a cathode, a diode, an anode, one terminal pin connected to the cathode and one terminal pin connected to the anode, a p-layer of semiconductor, an active region, an n-layer of semiconductor, a substrate and a transparent plastic case. The plastic case is transparent so that a photodetector outside the LED case can detect the arrival times of photons

emitted by the LED. In an embodiment, photodiode **944** in FIG. 9A absorbs photons emitted by LED **1002**. In an embodiment, phototransistor **954** in FIG. 9B absorbs photons emitted by LED **1002**.

The emission times of the photons emitted by the LED experimentally obey the energy-time form of the Heisenberg uncertainty principle. The energy-time form of the Heisenberg uncertainty principle contributes to the non-determinism of random noise generator **142** because the photon emission times are unpredictable due to the uncertainty principle. In FIG. 9A and FIG. 9B, the arrival of photons are indicated by a squiggly curve with an arrow and $h\nu$ next to the curve. The detection of arrival times of photons is a non-deterministic process. Due to the uncertainty of photon emission, the arrival times of photons are quantum events.

In FIG. 9A, $h\nu$ refers to the energy of a photon that arrives and is absorbed at photodiode **944** where h is Planck's constant and ν is the frequency of the photon. In FIG. 9B, $h\nu$ refers to the energy of a photon that arrives and is absorbed at phototransistor **954**. In FIG. 9A, the p and n semiconductor layers are a part of a phototransistor **944**, which generates and amplifies electrical current, when photons (light) are absorbed by the phototransistor. In FIG. 9B, the p and n semiconductor layers are a part of a photodiode **954**, which absorbs photons that arrive at photodiode **954**.

A photodiode is a semiconductor device that converts light (photons) into electrical current, which is called a photocurrent. The photocurrent is generated when photons are absorbed in the photodiode. Photodiodes are similar to standard semiconductor diodes except that they may be either exposed or packaged with a window or optical fiber connection to allow light (photons) to reach the sensitive part of the device. A photodiode may use a PIN junction or a p-n junction to generate electrical current from the absorption of photons. In some embodiments, the photodiode may be a phototransistor.

A phototransistor is a semiconductor device comprised of three electrodes that are part of a bipolar junction transistor. Light or ultraviolet light activates this bipolar junction transistor. Illumination of the base generates carriers which supply the base signal while the base electrode is left floating. The emitter junction constitutes a diode, and transistor action amplifies the incident light inducing a signal current.

When one or more photons with high enough energy strikes the photodiode, it creates an electron-hole pair. This phenomena is a type of photoelectric effect. If the absorption occurs in the junction's depletion region, or one diffusion length away from the depletion region, these carriers (electron-hole pair) are attracted from the PIN or p-n junction by the built-in electric field of the depletion region. The electric field causes holes to move toward the anode, and electrons to move toward the cathode; the movement of the holes and electrons creates a photocurrent. In some embodiments, the amount of photocurrent is an analog value, which can be digitized by an analog-to-digital converter. In some embodiments, the analog value is amplified before being digitized. The digitized value is what becomes the random noise. In some embodiments, a one-way hash function **948** in FIG. 9A or **958** in FIG. 9B, may also be applied to post-process the random noise to produce the noise $r_1 r_2 \dots r_p$ used by processes 1, 3, 5 6 and 7. In some embodiments, a one-way hash function may be applied to the random noise to produce key(s) $k_1 k_2 \dots k_m$, used by processes 3 and 6.

In an embodiment, the sampling of the digitized photocurrent values may be converted to threshold times as follows. A photocurrent threshold θ is selected as a sampling param-

25

eter. If a digitized photocurrent value i_1 is above θ at time t_1 , then t_1 is recorded as a threshold time. If the next digitized photocurrent value i_2 above θ occurs at time t_2 , then t_2 is recorded as the next threshold time. If the next digitized value i_3 above θ occurs at time t_3 , then t_3 is recorded as the next threshold time.

After three consecutive threshold times are recorded, these three times can determine a bit value as follows. If $t_2 - t_1 > t_3 - t_2$, then random noise generator produces a 1 bit. If $t_2 - t_1 < t_3 - t_2$, then random noise generator produces a 0 bit. If $t_2 - t_1 = t_3 - t_2$, then no noise information is produced. To generate the next bit, random noise generator **942** or **952** continues the same sampling steps as before and three new threshold times are produced and compared.

In an alternative sampling method, a sample mean μ is established for the photocurrent, when it is illuminated with photons. In some embodiments, the sampling method is implemented as follows. Let i_1 be the photocurrent value sampled at the first sampling time. i_1 is compared to p . E is selected as a parameter in the sampling method that is much smaller number than μ . If i_1 is greater than $\mu + E$, then a 1 bit is produced by the random noise generator **942** or **952**. If i_1 is less than $\mu - E$, then a 0 bit is produced by random noise generator **942** or **952**. If i_1 is in the interval $[\mu - E, \mu + E]$, then NO bit is produced by random noise generator **942** or **952**.

Let i_2 be the photocurrent value sampled at the next sampling time. i_2 is compared to u . If i_2 is greater than $\mu + E$, then a 1 bit is produced by the random noise generator **942** or **952**. If i_2 is less than $\mu - E$, then a 0 bit is produced by the random noise generator **942** or **952**. If i_2 is in the interval $[\mu - E, \mu + E]$, then NO bit is produced by the random noise generator **942** or **952**. This alternative sampling method continues in the same way with photocurrent values i_3 , i_4 , and so on. In some embodiments, the parameter E is selected as zero instead of a small positive number relative to p .

Some alternative hardware embodiments of non-deterministic generator **128** (FIG. 1B) are described below. In some embodiments that utilize non-determinism to produce random noise, a semitransparent mirror may be used. In some embodiments, the mirror contains quartz (glass). The photons that hit the mirror may take two or more paths in space. In one embodiment, if the photon is reflected, then the random noise generator creates the bit value $b \in \{0, 1\}$; if the photon is transmitted, then the random noise generator creates the other bit value $1 - b$. In another embodiment, the spin of an electron may be sampled to generate the next non-deterministic bit. In still another embodiment of a random noise generator, a protein, composed of amino acids, spanning a cell membrane or artificial membrane, that has two or more conformations can be used to detect non-determinism: the protein conformation sampled may be used to generate a random noise value in $\{0, \dots, n-1\}$ where the protein has n distinct conformations. In an alternative embodiment, one or more rhodopsin proteins could be used to detect the arrival times $t_1 < t_2 < t_3$ of photons and the differences of arrival times ($t_2 - t_1 > t_3 - t_2$ versus $t_2 - t_1 < t_3 - t_2$) could generate non-deterministic bits that produce random noise.

In some embodiments, the seek time of a hard drive can be used as random noise values as the air turbulence in the hard drive affects the seek time in a non-deterministic manner. In some embodiments, local atmospheric noise can be used as a source of random noise. For example, the air pressure, the humidity or the wind direction could be used. In other embodiments, the local sampling of smells based on particular molecules could also be used as a source of random noise.

26

In some embodiments, a Geiger counter may be used to sample non-determinism and generate random noise. In these embodiments, the unpredictability is due to radioactive decay rather than photon emission, arrivals and detection.

10.6 Deterministic Generators

In an embodiment, a deterministic generator **962** (FIG. 9C) is implemented with a machine. In an embodiment, machine 1 generates noise **972** as follows. Φ is one-way hash function with digest size d and is executed with one-way hash instructions **964**. In some embodiments, Ψ is a one-way hash function with digest size at least p bits (noise size) and is executed with one-way hash instructions **968**. In some embodiments, if p is greater than digest size of Ψ , then the generator update steps in machine 1 may be called more than once to generate enough noise.

In some embodiments, Φ and Ψ are the same one-way hash functions. In other embodiments, Φ and Ψ are different one-way hash functions. In an embodiment, Φ is one-way hash function SHA-512 and Ψ is one-way hash function Keccak. In another embodiment, Φ is one-way hash function Keccak and Ψ is one-way hash function SHA-512.

In an embodiment, the i th generator $\Delta(i)$ is composed of N bits and updated with generator update instructions **966**. The N bits of $\Delta(i)$ are represented as $\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,N-1}$ where each bit $\Delta_{i,j}$ is a 0 or 1. In an embodiment, generator update instructions **966** are executed according to the following two steps described in machine 1:

Update $(\Delta_{i+1,0} \Delta_{i+1,1} \dots \Delta_{i+1,d-1}) = \Phi(\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,d-1})$
Update $\Delta_{i+1,j} = \Delta_{i,j}$ for each j satisfying $d \leq j \leq N-1$

In an embodiment, the size of the deterministic generator N may be 1024. In another embodiment, N may be fifty thousand. In another embodiment, N may be ten billion.

In an embodiment, one-way hash instructions **964** are performed by processor system **258** (FIG. 1B). In an embodiment, one-way hash instructions **968** are performed by processor system **258** (FIG. 1B). In an embodiment, generator update instructions **966** are performed by processor system **258** (FIG. 1B). In an embodiment, memory system **256** stores one-way hash instructions **964**, one-way hash instructions **968** and generator update instructions **966**.

In an embodiment, the instructions that execute machine 1 and help execute deterministic generator **962** may be expressed in the C programming language before compilation. In an embodiment, the instructions that execute machine 1 and help execute deterministic generator **962** may be expressed in the native machine instructions of processor system **258**. In an embodiment, the instructions that execute machine 1 may be implemented as an ASIC, which is part of processor system **258**.

Machine 1. Generating Noise with a Machine

0th generator state $\Delta(0) = \Delta_{0,0} \dots \Delta_{0,N-1}$.

Initialize $i = 0$

while(hiding process **122** requests more noise)

{
Update $(\Delta_{i+1,0} \Delta_{i+1,1} \dots \Delta_{i+1,d-1}) = \Phi(\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,d-1})$
Update $\Delta_{i+1,j} = \Delta_{i,j}$ for each j satisfying $d \leq j \leq N-1$
Increment i
Generate noise **972** $r_1 r_2 \dots r_p$ by executing one-way hash Ψ instructions **968** on generator state $\Delta(i)$ as input to Ψ , where noise $r_1 r_2 \dots r_p$ is the first p bits of hash output $\Phi(\Delta_{i,0} \dots \Delta_{i,N-1})$.
}

In an embodiment, machine 2 generates key(s) **970** as follows. Φ is one-way hash function with digest size d and is executed with one-way hash instructions **964**. In some

embodiment, Ψ is a one-way hash function with digest size at least m bits (size of one or more keys) and is executed with one-way hash instructions **968**. In some embodiments, if m is greater than digest size of Ψ , then the generator update steps in machine 2 may be called more than once to generate enough keys.

In some embodiments, Φ and Ψ are the same one-way hash functions. In other embodiments, Φ and Ψ are different one-way hash functions. In an embodiment, Φ is one-way hash function SHA-512 and Ψ is one-way hash function Keccak. In another embodiment, Φ is one-way hash function Keccak and Ψ is one-way hash function SHA-512.

In an embodiment, the i th generator $\Delta(i)$ is composed of N bits and updated with generator update instructions **966**. The N bits of $\Delta(i)$ are represented as $\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,N-1}$ where each bit $\Delta_{i,j}$ is a 0 or 1. In an embodiment, generator update instructions **966** are executed according to the following two steps described in machine 2:

Update $(\Delta_{i+1,0} \Delta_{i+1,1} \dots \Delta_{i+1,d-1}) = \Phi(\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,d-1})$
Update $\Delta_{i+1,j} = \Delta_{i,j}$ for each j satisfying $d \leq j \leq N-1$

In an embodiment, the size of the deterministic generator N may be 1024. In another embodiment, N may be fifty thousand. In another embodiment, N may be ten billion.

In an embodiment, one-way hash instructions **964** are performed by processor system **258** (FIG. 1B). In an embodiment, one-way hash instructions **968** are performed by processor system **258** (FIG. 1B). In an embodiment, generator update instructions **966** are performed by processor system **258** (FIG. 1B). In an embodiment, memory system **256** stores one-way hash instructions **964**, one-way hash instructions **968** and generator update instructions **966**.

In an embodiment, the instructions that execute machine 2 and help execute deterministic generator **962** may be expressed in the C programming language before compilation. In an embodiment, the instructions that execute machine 2 and help execute deterministic generator **962** may be expressed in the native machine instructions of processor system **258**. In an embodiment, the instructions that execute machine 2 may be implemented as an ASIC, which is part of processor system **258**. In an embodiment, memory system **956** may store one or more keys **970**.

Machine 2. Generating One or more Keys with a Machine
0th generator state $\Delta(0) = \Delta_{0,0} \dots \Delta_{0,N-1}$.

Initialize $i=0$

while(hiding process **122** requests more key(s))

{

Update generator $(\Delta_{i+1,0} \Delta_{i+1,1} \dots \Delta_{i+1,d-1}) = \Phi(\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,d-1})$.

Update generator $\Delta_{i+1,j} = \Delta_{i,j}$ for each j satisfying $d \leq j \leq N-1$

Increment i

Generate key(s) **970** $k_1 k_2 \dots k_m$ by executing one-way hash V instructions **968** on generator state $\Delta(i)$ as input to Ψ , where $k_1 k_2 \dots k_m$ is the first m bits of hash output $\Phi(\Delta_{i,0} \dots \Delta_{i,N-1})$.

}

10.7 Scatter Map Hiding

A scatter map is a function that permutes data (information) to a sequence of distinct locations inside the random noise. To formally define a scatter map, the location space is defined first.

Definition 1.

Let $m, n \in \mathbb{N}$, where $m \leq n$. The set $\mathcal{L}_{m,n} = \{(l_1, l_2 \dots l_m) \in \{1, 2, \dots, n\}^m : l_j \neq l_k \text{ whenever } j \neq k\}$ is called an (m, n) location space.

Remark 1.

The location space $\mathcal{L}_{m,n}$ has

$$\frac{n!}{(n-m)!}$$

elements.

Definition 2.

Given a location element $(l_1, l_2 \dots l_m) \in \mathcal{L}_{m,n}$, the noise locations with respect to $(l_1, l_2 \dots l_m)$ are denoted as $\mathcal{N}(l_1, l_2 \dots l_m) = \{1, 2, \dots, n\} - \{l_i : 1 \leq i \leq m\}$.

Definition 3.

An (m, n) scatter map is an element $\pi = (l_1, l_2 \dots l_m) \in \mathcal{L}_{m,n}$ such that $\pi: \{0, 1\}^m \times \{0, 1\}^{n-m} \rightarrow \{0, 1\}^n$ and $\pi(d_1, \dots, d_m, r_1, r_2 \dots r_{n-m}) = (s_1, \dots, s_n)$ where the hiding locations s_i are selected as follows. Set $S_{l_i} = d_i$ $s_{l_1} = d_1 \dots s_{l_m} = d_m$. For the noise locations, set $s_{i_k} = r_k$ for the smallest subscript $i_k \in \mathcal{N}(\pi)$. Set $s_{i_k} = r_k$ for the k th smallest subscript $i_k \in \mathcal{N}(\pi)$.

Definition 3 describes how the scatter map selects the hiding locations of the parts of the key or data hidden in the noise. Furthermore, the scatter map process stores the noise in the remaining locations that do not contain parts of the key or data. Before the scatter map process begins, it is assumed that an element $\pi \in \mathcal{L}_{m,n}$ is randomly selected with a uniform distribution and Alice and Bob already have secret scatter map $\pi = (l_1, l_2 \dots l_m)$.

Hiding Process 1. Scatter Map Process Hides Data before Transmitting

Alice retrieves data $d_1 d_2 \dots d_m$.

Alice generates noise $r_1 r_2 \dots r_p$ with her random noise generator.

Per definition 3, Alice uses scatter map π to store her data $s_{l_1} = d_1 \dots s_{l_m} = d_m$.

Per definition 3, Alice stores the noise in the noise (unoccupied) locations of

$S = (s_1 \dots s_n)$ so that the data $d_1 d_2 \dots d_m$ is hidden in the noise.

Alice sends S to Bob.

Bob receives S .

Bob uses scatter map T to extract data $d_1 \dots d_m$ from S .

In an embodiment of process 1, scatter map π is executed by scatter map instructions **130** (FIG. 1B) and these instructions follow definition 3. In FIG. 2B, processor system **258** executes scatter map process instructions **130** during the step Alice uses scatter map w to store her data $s_{l_i} = d_i \dots s_{l_m} = d_m$. In an embodiment, scatter map process instructions **130** are stored in memory system **256** (FIG. 2B). In FIG. 2B, processor system **258** executes scatter map process instructions **130** during the step Alice stores the noise in the noise (unoccupied) locations of $S = (s_1 \dots s_n)$ so that the data $d_1 d_2 \dots d_m$ is hidden in the noise.

In an embodiment of process 1, output system **252** in FIG. 2B is used during the step Alice sends S to Bob. Output system **252** is part of sending machine **102** in FIG. 1A. In an embodiment of process 1, input system **254** in FIG. 2B is used during the step Bob receives S . Input system **254** is a part of receiving machine **112** in FIG. 1A.

In an alternative embodiment, Alice (first party) hides data inside noise so that she can protect its confidentiality and retrieve it at a later time. In an embodiment, hidden data **186** of FIG. 1E is stored on memory system that is served by the Internet cloud **212** of computers **210, 214, 216, 218, 220**.

Hiding Process 2. Scatter Map Process Hides Data before Storing

Alice retrieves data $d_1 d_2 \dots d_m$.

Alice generates noise $r_1 r_2 \dots r_p$ with her random noise generator.

Per definition 3, Alice uses scatter map m to store her data

$s_i = d_1 \dots s_{l_m} = d_m$.

Per definition 3, Alice stores the noise in the noise (unoccupied) locations of $S = (s_1 \dots s_n)$ so that the data $d_1 d_2 \dots d_m$ is hidden in the noise.

Alice sends S to memory.

system

In hiding process 2, storing machine **180**, as shown in FIG. **1E**, executes hiding process **184** to hide data **182** and stores the hidden data **186** in memory system **188**. In an embodiment, memory system **188** is implemented in hardware memory device **202** of FIG. **2A**. Memory device **202** is also shown in FIG. **3A**. In an embodiment, memory system **188** is part of mobile phone **400** in FIG. **400**. In an embodiment, hidden data **186** is stored in memory system **188**, which is implemented in the Internet cloud **212** of computers **210, 214, 216, 218, 220**, as shown in FIG. **2A**.

Hiding Process 3. Scatter Map Process Hides One or More Keys

Alice generates one or more keys $k_1 k_2 \dots k_m$ with her random noise generator and random noise $r_1 r_2 \dots r_p$.

Per definition 3, Alice stores one or more keys $s_{l_i} = k_1 \dots s_{l_m} = k_m$ using scatter map π .

Per definition 3, Alice stores the noise $r_1 r_2 \dots r_p$ in the noise (unoccupied) locations of $S = (s_1 \dots s_n)$ so that the one or more keys $k_1 k_2 \dots k_m$ are hidden in the noise.

Alice sends S to Bob.

Bob receives S .

Bob uses scatter map π to extract one or more keys $k_1 \dots k_m$ from S .

In an embodiment of process 3, scatter map π is executed by scatter map instructions **130** (FIG. **1B**) and these instructions follow definition 3. In FIG. **2B**, processor system **258** executes scatter map process instructions **130** during the step Alice stores one or more keys $s_{l_i} = k_1 \dots s_{l_m} = k_m$ using scatter map π . In an embodiment, scatter map process instructions **130** are stored in memory system **256** (FIG. **2B**). In FIG. **2B**, processor system **258** executes scatter map process instructions **130** during the step Alice stores the noise $r_1 r_2 \dots r_p$ in the noise (unoccupied) locations of $S = (s_1 \dots s_n)$ so that the one or more keys $k_1 k_2 \dots k_m$ are hidden in the noise. In FIG. **2B**, processor system **258** executes scatter map process instructions **130** during the step Bob uses scatter map m to extract one or more keys $k_1 \dots k_m$ from S .

In an embodiment of process 3, output system **252** is used during the step Alice sends S to Bob. Output system **252** is part of sending machine **102** in FIG. **1A**. In an embodiment of process 3, input system **254** is used during the step Bob receives S . Input system **254** is a part of receiving machine **112** in FIG. **1A**.

When the scatter size is n , process 1 takes n steps to hide the data inside the noise. When the scatter size is n , process 3 takes n steps to hide one or more keys inside the noise. When the bit-rate of a random noise generator is x bits per second, then a transmission with scatter size x bits is practical. When $x=10,000$, a key size of 2000 bits and noise size of 8000 bits is feasible. When $x=20,000$, a data size of 5000 bits and noise size of 1500 bits is feasible. In some applications, Alice and Bob may also establish the key size or data size m as a shared secret, where m is not disclosed to Eve.

In the interests of being conservative about the security, the mathematical analysis in section 10.11 assumes that Eve knows the data or key size m . For applications where Eve doesn't know m , the security will be stronger than the results obtained in the upcoming sections.

10.8 Multiple Hidings of Data Transmissions or Storings

This section analyzes the mathematics of when a scatter map is safest to reuse for multiple, hidings of data transmissions or hidings of data stored. Suppose that scatter map $\pi \in \mathcal{L}_{m,n}$ is established with Alice and Bob, according to a uniform probability distribution and adversary Eve has no information about π . Before Eve sees Alice's first scatter storage or first scatter transmission from Alice to Bob, from Eve's perspective, the probability

$$P(\pi = (l_1, l_2 \dots l_m)) = \frac{(n-m)!}{n!}$$

for each $(l_1, l_2 \dots l_m)$ in $\mathcal{L}_{m,n}$; in other words, Eve has zero information about π with respect to $\mathcal{L}_{m,n}$.

Rule 1. The Noise and Data (Machine Instruction) have the Same Bias

The noise and data (machine instruction) satisfy the same bias property if the probability distribution of the noise and the probability distribution of the data are the same. In an embodiment, the data is a machine instruction. In an embodiment, the noise is generated by non-deterministic generator **942** and has two outcomes (binary) and each outcome has probability $1/2$; and each indivisible unit of the data has two outcomes and each outcome has probability $1/2$.

In another embodiment the noise is generated by deterministic generator **962** and has two outcomes (binary) and the first outcome has probability $1/10$ and the second outcome has probability $9/10$; and each indivisible unit of the data has two outcomes and the first outcome for a data unit has probability $1/10$ and the second outcome has probability $9/10$.

In another embodiment the noise is generated by deterministic generator **962** and has three outcomes: the first outcome has probability $1/100$; the second outcome has probability $29/100$; and the third outcome has probability $70/100$. And the primitive units of data have three outcomes: the first outcome has probability $1/100$; the second outcome has probability $29/100$; and the third outcome has probability $70/100$.

Our next rule describes that prior history of outcomes has no influence on which outcome occurs next. In probability theory, this is sometimes called stochastic independence.

Rule 2. Stochastic Independence

History has no effect on the next event. Each outcome of is independent of the history. Let p_i be the probability of outcome o_i where there are m outcomes. A standard assumption of probability theory is that the sum of the probabilities of a finite number of mutually exclusive outcomes is 1 when these outcomes cover all possible outcomes. That is,

$$\sum_{i=1}^m p_i = 1.$$

Stochastic independence means that no correlation exists between previous or future outcomes. If the history of the prior $j-1$ outcomes is b_1, b_2, \dots, b_{j-1} , then the conditional probability of outcome o_i on the j th trial is still p_i , regardless

31

of this history. This is expressed in terms of the conditional probabilities: for every outcome o_i , $P(x_j=o_i|x_1=b_1, \dots, x_{j-1}=b_{j-1})=p_i$.

Process 4 shows how to generate biased noise that is unpredictable so that unencrypted data with a bias can still be effectively hidden. In hiding process 4, a line starting with `::` is a comment.

Hiding Process 4. A Biased Noise Generator

`:: The noise outcomes in \mathcal{O} are indexed as $\{o_1, o_2, \dots, o_m\}$, when there are m outcomes.
:: p_i is the probability of noise outcome o_i .
:: The interval $[0, 1)$ is subdivided into subintervals with rational endpoints.`

```
set i=1
set x=0
while(i≤m)
{
  set  $L_i=[x, x+p_i)$ 
  increment i
  update x to  $x+p_i$ 
}
```

set

$$\delta = \frac{1}{2} \min\{p_1, p_2, \dots, p_m\}$$

compute $n>0$ so that $2^{-n}<\delta$

use an unbiased, binary non-deterministic generator to sample n bits b_1, b_2, \dots, b_n

set

$$x = \sum_{i=1}^m b_i 2^{-i}$$

Generate noise outcome o_j such that x lies in interval L_j

In an embodiment, hiding process 4 can be implemented as a computing system (FIG. 2B) with input system 242, output system 240, memory system 246, and processor system 252, using non-deterministic generator 942 or non-deterministic generator 952. In an embodiment, hiding process 4 can be used to help implement random noise generator 128 in FIG. 1B. In an embodiment, hiding process 4 can be used to generate biased noise 990 in FIG. 9D. In an embodiment, the unbiased, binary non-deterministic generator in hiding process 4 is implemented with non-deterministic generator 942 of FIG. 9A or non-deterministic generator 952 of FIG. 9B.

Next, two more rules are stated whose purpose is to design embodiments that do not lead leak information to Eve. Section 10.13 shows some embodiments that authenticate the data or key(s) hidden in the noise. Embodiments that follow these rules help hinder Eve from actively sabotaging Alice and Bob to violate these rules.

Rule 3. New Noise and New Data

For each scattered transmission, described in process 1 or process 3, Alice creates new data $d_1 \dots d_m$ or creates a new key $k_1 \dots k_m$. Alice also creates new noise $r_1 \dots r_{n-m}$ from a non-deterministic generator that satisfies rule 1 (The Noise and Data have the Same Bias) and rule 2 (Stochastic Independence).

Rule 4. No Auxiliary Information

During the k th scattered transmission or storage, Eve only sees scattered $S(k)$; Eve receives no auxiliary information from Alice or Bob. Scattered $S(k)$ represents the key(s) or data hidden in the noise.

32

Theorem 1.

When Eve initially has zero information about π w.r.t. $\mathcal{L}_{m,n}$ and rules 3 and 4 hold, then Eve still has zero information about π after she observes scattered transmissions $S(1), S(2), \dots S(k)$.

In a proof of theorem 1, the following terminology is used. i lies in $\pi=(l_1, l_2 \dots l_m)$ if $i=l_j$ for some $1 \leq j \leq m$. Similarly, i lies outside π if $i \neq l_j$ for every $1 \leq j \leq m$. In this latter case, i is a noise location.

PROOF. Consider the i th bit location in the scattered transmission. Let $x_i(k)$ denote the i th bit observed by Eve during the k th scattered transmission $S(k)$. The scatter map π is established before the first transmission based on a uniform probability distribution; rule 3 implies the data generation and noise generation obey the two properties of no bias and history has no effect. These rules imply the conditional probabilities

$$P(x_i(k+1) = 1 | x_i(k) = b) = \frac{1}{2} = P(x_i(k+1) = 0 | x_i(k) = b)$$

hold for $b \in \{0, 1\}$, independent of whether i lies in π or i lies outside π . Rule 4 implies that if Eve's observation of $S(1), S(2), \dots S(k)$ enabled her to obtain some information, better than

$$P(\pi = (l_1, l_2 \dots l_m)) = \frac{(n-m)!}{n!},$$

about whether i lies in π or i lies outside π , then this would imply that the probability distribution of the noise is distinct from the probability distribution of the data, which is a contradiction. \square

Remark 2.

Theorem 1 is not precisely true if the probability distribution of the noise is distinct from the probability distribution of the data. In some embodiments, the probability distribution of the noise may be close enough to the probability distribution of the data so that Eve cannot obtain enough information to guess or figure out hiding locations. With this in mind, for a positive $\epsilon > 0$ and

$$\epsilon < \frac{1}{2},$$

we define what it means for the probability distribution of the noise to be ϵ -close to the probability distribution of the data.

Definition 4. Probability Distributions that are ϵ -Close

Suppose the noise outcomes and the data outcomes are binary. That is, each noise outcome can be represented with a bit and each data outcome can be represented with a bit. Let (p_0, p_1) be the noise distribution. This means that the probability of a noise bit being a 0 is p_0 and probability of a noise bit being 1 is p_1 , so $p_0+p_1=1$. Let (q_0, q_1) be the data distribution. This means that the probability of a data bit being a 0 is q_0 and probability of a data bit being 1 is q_1 . The probability distribution of the noise is ϵ -close to the probability distribution of the data if $|p_0-q_0|<\epsilon$ and $|p_1-q_1|<\epsilon$.

Suppose the noise has m distinct outcomes and the data has m distinct outcomes. Let (p_1, p_2, \dots, p_m) be the probability distribution of the noise, where

$$\sum_{i=1}^m p_i = 1.$$

Let (q_1, q_2, \dots, q_m) be the probability distribution of the data, where

$$\sum_{i=1}^m q_i = 1.$$

The probability distribution of the noise is ϵ -close to the probability distribution of the data if $|p_i - q_i| < \epsilon$ for every i such that $1 \leq i \leq m$.

In embodiments, remark 2 advises us not to let Alice violate rule 3: an example of what Alice should not do is send the same data or same key in multiple executions of process 1 or process 3 and the noise is randomly generated for each execution.

10.9 Effective Hiding

This section provides the intuition for effective hiding. Effective hiding occurs when Eve obtains no additional information about scatter map σ after Eve observes multiple hidden key or hidden data transmissions. Section 10.8 provides mathematical analysis of this intuition.

The effectiveness of the hiding depends upon the following observation. Even after Eve executes a search algorithm for the data (signal) in the noise, Eve's search algorithm does NOT know when it has found the key or the data because her search algorithm CANNOT distinguish the signal from the noise. This is illustrated by FIG. 5 and FIG. 6.

The pixel values in FIG. 5 and FIG. 6 that compose the secret are hidden in the noise of the visual image such that the probabilities of the pixel values satisfy the two randomness axioms. Suppose Eve performs a brute force search over all

$$\frac{n!}{(n-m)!}$$

possibilities for scatter map σ . Even if Eve's search method stumbles upon the correct sequence of locations, Eve's method has no basis for distinguishing the data from the noise because the key and noise probability distributions are equal. For FIG. 5, Eve does not have a terminating condition for halting with this sequence of bit locations hiding the key. For FIG. 6, Eve does not have a terminating condition for halting with this sequence of locations hiding the data.

In FIG. 7 and FIG. 8, Eve can obtain some locations of the hidden data or hidden key because the probability distribution of the secret (foreground) is not the same as the noise (background): Eve can determine the secret is located in a P shape, because the probability distribution of these secret pixels violates the randomness axioms.

10.10 Dynamic Hiding Locations

This section describes a further enhancement of our invention(s) and makes Eve's challenge far more difficult for her to capture or estimate the scatter map: the locations of where to hide the information, or data, or keys, or public

keys can dynamically change. This means the scatter map dynamically changes in a way so that Alice can extract the hidden information at a later time even though each block of data units are hidden at different locations. In terms of mathematics, the scatter map $T(k)$ dynamically changes as a function of the k th storing or k th transmission. Precisely, $\pi(j) \neq \pi(k)$ when $j \neq k$.

In an embodiment, Alice hides information in blocks of 4 data (information) units and the noise size is 60 data (information) units. In some embodiments a unit may represent a bit: there are two choices for each d_1 . In other embodiments, there may be 4 choices for each unit d_1 . In the first hiding of the data or information in the noise, Alice hides data unit d_1 at location **63**; Alice hides data unit d_2 at location **2**; Alice hides data unit d_3 at location **38**. As described in section, Alice stores the noise units in the remaining 60 locations.

In the second hiding of 4 data units d_4, d_5, d_6 , and d_7 , Alice hides data unit d_4 at location **28**; Alice hides data unit d_5 at location **51**; Alice hides data unit d_6 at location **46**; and Alice hides data unit d_7 at location **12**. The hiding locations in the first hiding are different from the hiding location in the second hiding. In other words, the enhancement described in machine 3 provides dynamic hiding locations.

Described below, machine 3 describes an embodiment of hiding locator machine **980** of FIG. 9D, that generates unpredictable hiding locations so that even if Eve knows scatter maps $\pi(1), \pi(2), \dots$, and all the way up to $\pi(k-1)$, Eve will not be able to guess or effectively predict scatter map $x(k)$.

Machine 3. Dynamic Hiding Locations

0th hiding locator $\Delta(0) = \Delta_{0,0} \dots \Delta_{0,N-1}$.

Initialize $i=0$

```
while(hiding process 122 requests more location(s))
{
    Update hiding locator  $(\Delta_{i+1,0} \Delta_{i+1,1} \dots \Delta_{i+1,d-1}) = \Phi(\Delta_{i,0} \Delta_{i,1} \dots \Delta_{i,d-1})$ .
    Update hiding locator  $\Delta_{i+j} = \Delta_{i,j}$  for each  $j$  satisfying  $d \leq j \leq N-1$ 
    Increment  $i$ 
    Generate scatter map locations 970  $l_1, l_2, \dots, l_m$  by
    executing one-way hash  $\Phi$  instructions 968 on hiding
    locator  $\Delta(i)$  as input to  $\Phi$ , where  $l_1 l_2 \dots l_m$  is the
    first  $b$   $m$  bits of hash output  $\Phi(\Delta_{i,0} \dots \Delta_{i,N-1})$ .
}
```

In a machine 3 embodiment of hiding locator machine **980**, $N=1024$ is the size of the hiding locator $\Delta(i)$, and $\Delta(i)$ is updated by hiding locator update instructions **986**. In another embodiment, N is fifty thousand. In another embodiment, N is ten billion.

In a machine 3 embodiment, $b=32$. Since 232 is greater than 4 billion, there are more than 4 billion possible hiding locations for a single hiding of data when $b=32$. In a machine 3 embodiment, $b=64$. Since 264 is greater than 10^{19} , there are more than 10^{19} possible hiding locations for a single hiding of data when $b=64$. In a machine 3 embodiment, $b=5000$.

In a machine 3 embodiment of hiding locator machine **980**, one-way hash instructions **984** are performed by processor system **258** (FIG. 1B). In an embodiment, one-way hash instructions **968** are performed by processor system **258** (FIG. 1B). In an embodiment, hiding locator update instructions **986** are performed by processor system **258** (FIG. 1B). In an embodiment, memory system **256** stores one-way hash instructions **964**, one-way hash instructions **968** and hiding locator update instructions **986**.

35

In an embodiment, the instructions that execute machine 3 and help execute hiding locator machine **980** may be expressed in the C programming language before compilation. In an embodiment, the instructions that execute machine 3 and help execute hiding locator machine **980** may be expressed in the native machine instructions of processor system **258**. In an embodiment, the instructions that execute machine 3 may be implemented as an ASIC, which is part of processor system **258**. In an embodiment, memory system **956** may store one or more dynamic hiding locations **982**.

10.11 Single Storage and Transmission Analysis

The size of the location space is significantly greater than the data or key size. Even for values of n as small as 30,

$$\frac{n!}{(n-m)!} \gg 2^m.$$

The uniform distribution of the noise and the data generation and a large enough noise size poses Eve with the challenge that even after seeing the transmission $S=(s_1 \dots s_n)$, she has almost no more information about the data or key(s), than before the creation of $k_1 k_2 \dots k_m$. The forthcoming analysis will make this notion of almost no more information more precise.

In some applications, Alice and Bob may also establish the data size m as a shared secret, where m is not disclosed to Eve. In the interests of being conservative about the security, it is assumed that Eve knows the data size m . For applications where Eve doesn't know m , the information security will be stronger than the results obtained in this section.

Processes 1 and 3 are analyzed with counting and asymptotic results that arise from the binomial distribution. First, some preliminary definitions are established.

For $0 \leq i \leq n$, define $E_{i,n} = \{r \in \{0, 1\}^n : \rho_1(r) = i\}$. When $n=4$, $E_{0,4} = \{0000\}$, $E_{1,4} = \{0001, 0010, 0100, 1000\}$, $E_{2,4} = \{0011, 0101, 0110, 1001, 1010, 1100\}$, $E_{3,4} = \{0111, 1011, 1101, 1110\}$ and $E_{4,4} = \{1111\}$. Note

$$|E_{k,n}| = \frac{n!}{(n-k)! k!} = \binom{n}{k}.$$

The expression—the i th element of $E_{k,n}$ —refers to ordering the set $E_{k,n}$ according to an increasing sequence of natural numbers that each binary string represents and selecting the i th element of this ordering. For example, the 3rd element of $E_{2,4}$ is 0110.

In FIG. 13, event $B_{i,j}$ refers to the i th data in $E_{j,m}$. Event R_i refers to the set of random noise elements which have i ones, and the noise size is $\rho = n - m$. Event A_i refers to a scatter $(s_1 \dots s_n)$ which contains i ones.

Equation 10.1 follows from the independence of events R_k and $B_{l,j}$.

$$P(R_k \cap B_{l,j}) = P(R_k) \cap P(B_{l,j}) \quad (10.1)$$

whenever $0 \leq k \leq \rho$ and $0 \leq j \leq m$ and $1 \leq l \leq \binom{m}{j}$.

Equation 10.2 follows from the definitions in FIG. 13 $\eta_1(s_1 \dots s_n) = \eta_1(r_1 \dots r_\rho) + \eta_1(k_1 \dots k_m)$; and the meaning of conditional probability.

$$P(A_k | B_{l,j}) = P(R_{k-j}) = \binom{\rho}{k-j} 2^{-\rho} \quad (10.2)$$

36

whenever $0 \leq j \leq \min\{k, m\}$ and

$$1 \leq l \leq \binom{m}{j}.$$

A finite sample space and

$$P\left(\bigcup_{j=0}^m \bigcup_{l=1}^{|E_{j,m}|} B_{l,j}\right) = 1$$

imply that each event

$$A_k \subset \bigcup_{j=0}^m \bigcup_{l=1}^{|E_{j,m}|} B_{l,j}.$$

Furthermore, $B_{l_1,j_1} \cap B_{l_2,j_2} = \emptyset$ whenever $l_1 \neq l_2$ or $j_1 \neq j_2$ such that $0 \leq j_1, j_2 \leq m$ and $1 \leq l_1 \leq |E_{j_1,m}|$ and $1 \leq l_2 \leq |E_{j_2,m}|$. Thus, Bayes Law is applicable. Equation 10.3 follows from Bayes Law and the derivation below 10.3.

$$P(B_{l,j} | A_k) = \frac{\binom{\rho}{k-j}}{\sum_{b=0}^{\min\{k,m\}} \binom{m}{b} \binom{\rho}{k-b}} \quad (10.3)$$

whenever $0 \leq j \leq \min\{k, m\}$ and $1 \leq l \leq \binom{m}{j}$. The mathematical steps that establish equation 10.3 are shown below.

$$\begin{aligned} P(B_{l,j} | A_k) &= \frac{P(B_{l,j})P(A_k | B_{l,j})}{\sum_{b=0}^{\min\{k,m\}} \sum_{a=1}^{|E_{b,m}|} P(B_{a,b})P(A_k | B_{a,b})} \\ &= \frac{P(A_k | B_{l,j})}{\sum_{b=0}^{\min\{k,m\}} \sum_{a=1}^{|E_{b,m}|} P(A_k | B_{a,b})} \\ &= \frac{\binom{\rho}{k-j} 2^{-\rho}}{\sum_{b=0}^{\min\{k,m\}} |E_{b,m}| \binom{\rho}{k-b} 2^{-\rho}}. \end{aligned}$$

Definition 5. Let c be a positive integer. $f: \mathbb{N} \rightarrow \mathbb{N}$ is called a binomial c -standard deviations function if there exists $N \in \mathbb{N}$ such that whenever $\rho \geq N$, then

$$\left| f(\rho) - \frac{\rho}{2} \right| \leq c \frac{\sqrt{\rho}}{2}$$

Define the function

$$h_c(\rho) = \max\left\{0, \frac{\rho}{2} - \left\lfloor c \frac{\sqrt{\rho}}{2} \right\rfloor\right\}.$$

Then h_c is a binomial c -standard deviations function. Lemmas 2 and 3 may be part of the binomial distribution folklore; for the sake of completeness, they are proven below.

Lemma 2.

Let $k: \mathbb{N} \rightarrow \mathbb{N}$ be a binomial c -standard deviations function. Then

$$\lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-1}{k(\rho)}}{\binom{k(\rho)}{k(\rho)}} = 1.$$

PROOF. A simple calculation shows that

$$\frac{\binom{k(\rho)-1}{k(\rho)}}{\binom{k(\rho)}{k(\rho)}} = \frac{k(\rho)}{\rho - k(\rho) + 1}.$$

Since $k(\rho)$ is a binomial c -standard deviations function,

$$\frac{\rho}{2} - \frac{c\sqrt{\rho}}{2} \leq k(\rho) \leq \frac{\rho}{2} + \frac{c\sqrt{\rho}}{2}.$$

This implies

$$\frac{\rho}{2} + \frac{c\sqrt{\rho}}{2} + 1 \geq \rho - k(\rho) + 1 \geq \frac{\rho}{2} - \frac{c\sqrt{\rho}}{2}.$$

Thus,

$$\frac{\frac{\rho}{2} - \frac{c\sqrt{\rho}}{2}}{\frac{\rho}{2} + \frac{c\sqrt{\rho}}{2} + 1} \leq \frac{\binom{k(\rho)-1}{k(\rho)}}{\binom{k(\rho)}{k(\rho)}} \leq \frac{\frac{\rho}{2} + \frac{c\sqrt{\rho}}{2}}{\frac{\rho}{2} - \frac{c\sqrt{\rho}}{2}}.$$

Since

$$\lim_{\rho \rightarrow \infty} \frac{\frac{\rho}{2} - \frac{c\sqrt{\rho}}{2}}{\frac{\rho}{2} + \frac{c\sqrt{\rho}}{2} + 1} = 1 = \lim_{\rho \rightarrow \infty} \frac{\frac{\rho}{2} + \frac{c\sqrt{\rho}}{2}}{\frac{\rho}{2} - \frac{c\sqrt{\rho}}{2}},$$

apply the squeeze theorem to equation 10.4. \square

The work from lemma 2 helps prove lemma 3. Lemma 3 helps prove that equation 10.3 converges to 2^{-m} when $k(\rho)$ is a binomial c -standard deviations function.

Lemma 3.

Fix $m \in \mathbb{N}$. Let $k: \mathbb{N} \rightarrow \mathbb{N}$ be a binomial c -standard deviations function. For any b, j such that $0 \leq b, j \leq m$, then

$$\lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-j}{k(\rho)-b}}{\binom{k(\rho)-b}{k(\rho)-j}} = 1.$$

PROOF. Using a similar computation to equation 10.4 inside of $c+1$ standard deviations instead of c , then ρ can be made large enough so that $k(\rho)-b$ and $k(\rho)-j$ lie within $c+1$ standard deviations so that

$$\lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-i-1}{k(\rho)-i}}{\binom{k(\rho)-i}{k(\rho)-i}} = 1$$

where $0 \leq i \leq m$. W.L.O.G., suppose $j < b$. Thus,

$$\lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-j}{k(\rho)-b}}{\binom{k(\rho)-b}{k(\rho)-j}} = \lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-j}{k(\rho)-(j+1)}}{\binom{k(\rho)-(j+1)}{k(\rho)-(j+2)}} \cdots \lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-(b-1)}{k(\rho)-b}}{\binom{k(\rho)-b}{k(\rho)-b}} = 1$$

Theorem 4.

Fix data size $m \in \mathbb{N}$. Let $c \in \mathbb{N}$. Let $k: \mathbb{N} \rightarrow \mathbb{N}$ be a binomial c -standard deviations function. Then

$$\lim_{\rho \rightarrow \infty} P(B_{l,j} | A_{k(\rho)}) = 2^{-m}.$$

PROOF

$$\lim_{\rho \rightarrow \infty} P(B_{l,j} | A_{k(\rho)}) = \lim_{\rho \rightarrow \infty} \left[\frac{\sum_{b=0}^{\min\{k(\rho), m\}} \binom{m}{b} k_{(\rho)-b}^{\rho}}{\binom{k(\rho)-j}{k(\rho)-j}} \right]^{-1}$$

from equation 10.3.

$$= \left[\sum_{b=0}^m \binom{m}{b} k_{(\rho)-b}^{\rho} \lim_{\rho \rightarrow \infty} \frac{\binom{k(\rho)-b}{k(\rho)-j}}{\binom{k(\rho)-j}{k(\rho)-j}} \right]^{-1}$$

(10.4) since m is fixed and $\rho \rightarrow \infty$ implies $k(\rho) > m$.
 $= 2^{-m}$ from lemma 2.

Remark 3.

Theorem 4 is not true when $k(\rho)$ stays on or near the boundary of Pascal's triangle. Consider

$$\lim_{\rho \rightarrow \infty} \frac{\binom{\rho}{0}}{\binom{\rho}{1}} = 0$$

or

$$\lim_{\rho \rightarrow \infty} \frac{\binom{\rho}{1}}{\binom{\rho}{2}} = 0.$$

The math confirms common sense: namely, if Eve sees event A_0 , then Eve knows that Alice's data is all zeroes. A practical and large enough noise size enables process 1 or process 3 to effectively hide the data transmission so that outlier events such as A_0, A_1 do not occur in practice. For example, when $n=2048$, $P(A_0)=2^{-2048}$ and $P(A_1)=2^{-2037}$.

Definitions 6, 7 and theorems 5, 6 provide a basis for calculating how big the noise size should be in order to establish an extremely low probability that Eve will see outlier events such as A_0 .

Definition 6.

$f: \mathbb{N} \rightarrow \mathbb{N}$ is an binomial ϵ -tail function if there exists $N \in \mathbb{N}$ such that $n \geq N$ implies that

$$2^{-n} \left(\sum_{k=0}^{f(n)} \binom{n}{k} + \sum_{k=n-f(n)}^n \binom{n}{k} \right) < \epsilon.$$

The area under the standard normal curve from $-\infty$ to x is expressed as

$$\Phi(x) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^x e^{-\frac{1}{2}t^2} dt.$$

Theorem 5.

For each $c \in \mathbb{N}$, set $\epsilon_c = 4 \Phi(-c)$. The function

$$g_c(n) = \max \left\{ 0, \left\lfloor \frac{n}{2} - c \frac{\sqrt{n}}{2} \right\rfloor \right\}$$

is a binomial ϵ_c -tail function.

PROOF. This is an immediate consequence of the central limit theorem [51, 52], applied to the binomial distribution. Some details are provided.

Define

$$B_n(x) = 2^{-n} \sum_{k=0}^{\lfloor x \rfloor} \binom{n}{k}.$$

In [53], DeMoivre proved for each fixed x that

$$\lim_{n \rightarrow \infty} B_n \left(\frac{n}{2} + x \frac{\sqrt{n}}{2} \right) = \Phi(x).$$

Thus,

$$\lim_{n \rightarrow \infty} 2^{-n} \sum_{k=0}^{g_c(n)} \binom{n}{k} = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-c} e^{-\frac{1}{2}t^2} dt.$$

Now ϵ_c is four times the value of

$$\frac{1}{\sqrt{2\pi}} \int_{-\infty}^{-c} e^{-\frac{1}{2}t^2} dt,$$

which verifies that g_c is a binomial ϵ_c -tail function. \square

Example 1

This example provides some perspective on some ϵ -tails and Eve's conditional probabilities. For $n=2500$, the scatter mean μ is 1250 and the standard deviation

$$\sigma = \sqrt{\frac{2500}{2}} = 25.$$

Set $c=20$, so $\mu - c\sigma = 750$. A calculation shows that

$$2^{-2500} \sum_{j=0}^{750} \binom{2500}{j} < 10^{-91}.$$

For $n=4096$, the scatter mean is 2048 and the standard deviation $\sigma=32$. Set $c=50$ standard deviations, so $\mu - c\sigma = 448$. A calculation shows that

$$2^{-4096} \sum_{j=0}^{448} \binom{4096}{j} < 10^{-621}.$$

Some of Eve's conditional probabilities are calculated for $n=2500$ and data size $m=576$. The average number of 1's in a key is $\mu_{key}=288$ and the standard deviation $\sigma_{key}=12$.

A typical case is when $j=300$ and $k=1275$, which are both one standard deviation to the right of the data and scatter mean, respectively. When Eve's conditional probability equals 2^{-m} , the secrecy ratio is exactly 1. Using equation 10.3, a computer calculation shows that the secrecy ratio is

$$\frac{P(B_{i,300}|A_{1275})}{2^{-576}} \approx 1.576,$$

so $2^{-576} < P(B_{i,300}|A_{1275}) < 2^{-575}$.

A rare event is when $j=228$ and $k=1225$. That is, $j=228$ is five standard deviations to the left of μ_{key} and $k=1225$ is one standard deviation to the left of the scatter mean. A calculation shows that

$$\frac{P(B_{i,228}|A_{1225})}{2^{-576}} \approx 0.526.$$

Thus, $2^{-577} < P(B_{i,228}|A_{1225}) < 2^{-576}$.

An extremely rare event occurs when $j=228$ and $k=1125$. Event A_{1125} is 4 standard deviations to the left.

$$\frac{P(B_{i,228}|A_{1125})}{2^{-576}} \approx 3840.$$

Thus, $2^{-303} < P(B_{i,228}|A_{1125}) < 2^{-564}$. While a secrecy ratio of 3840 is quite skew, it still means that even if Eve sees a scatter transmission 4 standard deviations to the left, there is still a probability in the interval $[2^{-565}, 2^{-564}]$ of Alice's data element being the event $B_{i,228}$.

Even when Eve sees a highly skewed, scattered transmission and obtains some information about the current hidden data element, Eve's observation provides her with no information about the next data element hidden in a subsequent transmission. The secrecy ratio calculations in example 1 provide the motivation for definition 7.

Definition 7.

Let $\epsilon > 0$. Eve's conditional probabilities $P(B_{i,j}|A_{k(\rho)})$ are ϵ -close to perfect secrecy if there exists a binomial ϵ -tail function f such that for any function $k: \mathbb{N} \rightarrow \mathbb{N}$ satisfying $f(\rho) \leq k(\rho) \leq \rho - f(\rho)$, then

$$\lim_{\rho \rightarrow \infty} P(B_{i,j}|A_{k(\rho)}) = 2^{-m}.$$

Theorem 6.

For any $\epsilon > 0$, there exists $M \in \mathbb{N}$ such that $\epsilon_c < \epsilon$ for all $c \geq M$ and $c \in \mathbb{N}$. Furthermore, function g_c is a binomial ϵ_c -tail function that makes Eve's conditional probabilities $P(B_{i,j}|A_{k(\rho)}) \in \epsilon_c$ -close to perfect secrecy, where $g_c(\rho) \leq k(\rho) \leq \rho - g_c(\rho)$.

PROOF. Since

$$\lim_{x \rightarrow \infty} \Phi(-x) = 0,$$

there exists $M \in \mathbb{N}$ such that $\epsilon_c < \epsilon$ for all $c \geq M$. Recall that

$$h_c(\rho) = \max\left\{0, \frac{\rho}{2} - \left\lceil c \frac{\sqrt{\rho}}{2} \right\rceil\right\}.$$

For all $\rho \in \mathbb{N}$, $|g_c(\rho) - h_c(\rho) - h_c(\rho)| \leq 1$ and $g_c(4\rho^2) - h_c(4\rho^2) = 0$. This fact and h_c is a binomial c -standard deviations function together imply that lemma 3 and hence theorem 4 also hold for function g_c . That is,

$$\lim_{\rho \rightarrow \infty} P(B_{i,j}|A_{g_c(\rho)}) = 2^{-m}.$$

Whenever function k satisfies $g_c(\rho) \leq k(\rho) \leq \rho - g_c(\rho)$, this implies k is a binomial $c+1$ -standard deviations function. Thus, this theorem immediately follows from theorems 4, 5 and from definition 7.

10.12 Data Transformations

In some embodiments, the key or data may be transformed by the sender (Alice) before being scattered and subsequently transmitted to the receiver (Bob). In an embodiment, each bit of the key or the data may be transformed according to the map $\Phi: \{0, 1\} \rightarrow \{01, 10\}$ where $\Phi(0)=01$ and $\Phi(1)=10$. Suppose the data is $K=010010000$. Φ^{-1} denotes the inverse of Φ . The inverse of Φ is used by Bob to reconstruct the data $d_1 d_2 \dots d_m$ from the transformed data $t_1 t_2 \dots t_{2m}$ after Bob extracts $t_1 t_2 \dots t_{2m}$ from the scattered transmission received from Alice. Note that $\Phi^{-1}(01)=0$ and $\Phi^{-1}(10)=1$. In some embodiments, data transformation instructions **126**, shown in FIG. **1B**, compute map Φ and inverse map Φ^{-1} .

After applying Φ to each bit of K , the transformation is $\Phi(0) \Phi(1) \Phi(0) \Phi(0) \Phi(1) (0)+(0)+(0) \neq (0)=01 10 01 01 10 01 01 01 01$. After this transformation by Φ , each of these 18 bits is scattered inside random noise. Suppose K is scattered inside of 130 bits of noise, then the location space will be $\mathcal{L}_{18,148}$. A scatter map π in $\mathcal{L}_{18,148}$ has 18 locations. That is, $\pi=(l_1, l_2, \dots, l_{18})$ and each l_i satisfies $1 \leq l_i \leq 148$.

FIG. **12** shows an embodiment with $l_1=37$, $l_2=29$, $l_{17}=4$ and $l_{18}=147$. In other embodiments, when the Φ transformation is used, for a data size m , a scatter map is selected from $\mathcal{L}_{2m,n}$ to hide the transformed $2m$ bits inside of $n-2m$ bits of random noise. In an embodiment, the data size may be $m=32$. In another embodiment, the data size may be $m=128$. In another embodiment, the data size may be

$m=1000$. In another embodiment, the data size may be $m=4096$. In another embodiment, the data size may be $m=512000$.

In alternative embodiments, the map $\Phi: \{0, 1\} \rightarrow \{01, 10\}$ where $\Phi(0)=10$ and $\Phi(1)=01$, may be used to transform the data before scattering (hiding) the data inside the noise. In an embodiment, the map Φ transforms the 16 bits of data 0100 1110 1001 0110 to a 32-bit transformation 10011010 01010110 01101001 10010110, before this 32-bit transformation is scattered by the sender (Alice) inside of random noise. After Bob extracts the transformed data 10011010 01010110 01101001 10010110 from the scattered transmission, Bob applies the inverse of Φ to each substring of two bits. For the first two bits, $\Phi^{-1}(10)=0$, so $d_1=0$. For bits **3** and **4**, Bob computes $\Phi^{-1}(01)=1$, so Bob reconstructs $d_2=1$. For bits **5** and **6**, Bob computes $\Phi^{-1}(10)=0$, so his third reconstructed data bit $d_3=0$. Bob continues this reconstruction of the 16th bit of data with bits **31** and **32** and computes $\Phi^{-1}(10)=0$, and reconstructs bit $d_{16}=0$. In some embodiments, data transformation instructions **126**, shown in FIG. **1B**, compute map Φ and inverse map Φ^{-1} .

Before the scatter map process using a data transformation is started, an element $\pi \in \mathcal{L}_{2m,n}$ is randomly selected and securely distributed to Alice and Bob. Note $2m < n$.

Hiding Process 5. Scatter Map Method using a Data Transformation Φ

Alice and Bob already have secret scatter map $\pi=(l_1, l_2 \dots l_{2m})$.

A Alice generates data $d_1 d_2 \dots d_m$ and noise $r_1 r_2 \dots r_{n-2m}$ from her random noise generator.

B Alice transforms data $d_1 d_2 \dots d_m$ to $t_1 t_2 \dots t_{2m}$ with transformation Φ .

C According to definition 3, Alice uses π to set $s_{l_i}=t_1 \dots s_{l_{2m}}=t_{2m}$.

D Per definition 3, Alice stores the noise at noise (unoccupied) locations in $S=(s_1 \dots s_n)$ so that the transformed data is hidden inside the noise.

E Alice sends $S=(s_1 \dots s_n)$ to Bob.

F Bob uses scatter map π to extract the transformed data $t_1 \dots t_{2m}$ from S .

G Bob applies the inverse of Φ to $t_1 \dots t_{2m}$ and reads data $d_1 d_2 \dots d_m$.

10.13 Hiding Data Elements with Authentication

It is assumed that Alice and Bob have previously established secret scatter map $\sigma=(l_1, l_2 \dots l_m)$ and authentication key k . In some embodiments, Alice and Bob may establish scatter map σ and authentication key k with a Diffie-Hellman-Merkle exchange [54, 55], where their public keys are signed in a secure computing or private manufacturing environment; alternatively, in other embodiments, Alice and Bob may establish σ and k via a different channel or in the same physical location by a face to face exchange or using a physical delivery by a mutually trusted courier.

Let h_k denote an MAC (e.g., HMAC or [65]) function which will be used to authenticate the scattered transmission. The use of h , helps hinder the following attack by Eve. An active Eve could flip a bit at bit location l in the scattered transmission. If no authentication occurs on the noise and the hidden key bits, then upon Alice resending a scattered transmission due to Alice and Bob not arriving at the same session key secret, Eve gains information that l lies in σ . If the scattered transmission S is not authenticated, Eve's manipulation of the bits in S helps her violate rule 4.

Hiding Process 6. Hiding One or More Keys with Authentication

Alice's random generator creates one or more keys $k_1 k_2 \dots k_m$ and random noise $r_1 r_2 \dots r_p$.
 Per definition 3, Alice uses scatter map σ to set $s_i = k_1 \dots s_{l_m} = k_m$.
 Alice stores the noise $r_1 r_2 \dots r_p$ at noise (unoccupied) locations in $S = (s_1 \dots s_n)$ so that her one or more keys $k_1 k_2 \dots k_m$ are hidden inside the noise.
 Alice sends S and $h_k(S)$ to Bob.
 Bob receives S' and $h_k(S)$ from Alice. Bob computes $h(S')$ and checks it against $h_k(S)$.
 If $h_k(S')$ is valid, Bob uses scatter map σ to extract one or more keys $k_1 \dots k_m$ from S ;
 else Bob rejects S' and asks Alice to resend S .
 In some embodiments of process 6, scatter map σ is executed by scatter map instructions 130 (FIG. 1B) and these instructions follow definition 3. In some embodiments of process 6, Alice's random noise generator is implemented with random noise generator 128 (FIG. 1B). In some embodiments of process 6, Alice's random noise generator uses a light emitting diode as shown in FIGS. 9 and 10. In some embodiments of process 6, the key $k_1 \dots k_m$ has no bias and has the same probability distribution as the noise $r_1 r_2 \dots r_p$. In some embodiments of process 6, the data elements $e_1 \dots e_m$ are encrypted plaintext data instead of a key.

In some embodiments of process 6, the probability distribution of the data elements is biased and the probability distribution of the noise is biased. In preferred embodiments, the probability distribution of the data elements is the same as the probability distribution of the noise even though they are both biased. In some embodiments, the probability distribution of the data elements is almost the same the probability distribution of the noise. Almost the same probability distribution means that an average hacker that is eavesdropping on the hidden data transmissions would not be able to find where the data is being hidden after a seeing the hidden transmissions for a reasonable amount of time. In an embodiment, a reasonable amount of time is 3 months. In another embodiment, a reasonable amount of time is 1 year. In another embodiment, a reasonable amount of time is 5 years.

In other embodiments, Alice encrypts plaintext data d_1, \dots, d_m with a block or stream cipher before the encrypted data e_1, \dots, e_m is hidden in random noise; this is described in process 7 below.

Hiding Process 7. Hiding Encrypted Data Elements with Authentication

Alice's uses encryption algorithm \mathcal{E} and key K to encrypt data $M = d_1 d_2 \dots d_m$ as $\mathcal{E}(M, K) = e_1 e_2 \dots e_m$.

Per definition 3, Alice uses scatter map σ to set $s_{l_1} = e_1 \dots s_{l_m} = e_m$.

Alice's random noise generator creates noise $r_1 r_2 \dots r_p$. Alice stores the noise $r_1 r_2 \dots r_p$ at noise (unoccupied) locations in $S = (s_1 \dots s_n)$ so that the encrypted data $e_1 e_2 \dots e_m$ is hidden inside the noise.

Alice sends S and $h_K(S)$ to Bob.

Bob receives S' and $h_K(S)$ from Alice. Bob computes $h_K(S')$ and checks it against $h_K(S)$.

If $h_K(S')$ is valid, Bob uses scatter map σ to extract $e_1 \dots e_m$ from S and subsequently uses decryption algorithm \mathcal{D} and key K to decrypt $e_1 \dots e_m$ and obtain $d_1 \dots d_m$.
 else Bob rejects S' and asks Alice to resend S .

In some embodiments of process 7, catter map σ is executed by scatter map instructions 130 (FIG. 1B) and these instructions follow definition 3. In some embodiments of process 7, Alice's random noise generator is implemented with random noise generator 128 (FIG. 1B). In some

embodiments of process 7, Alice's random noise generator uses a light emitting diode as shown in FIGS. 9 and 10. In some embodiments, the encrypted data $e_1 \dots e_m$ has no bias and has the same probability distribution as the noise $r_1 r_2 \dots r_p$.

In some embodiments of process 7, encryption algorithm \mathcal{E} is the block cipher Serpent and is executed with cipher instructions 129 as shown in FIG. 1B. In some embodiments of process 7, encryption algorithm \mathcal{E} is the block cipher AES [39], and is executed with cipher instructions 129.

In some embodiments of process 7, encryption algorithm \mathcal{E} is a block cipher and also uses the cipher block chaining mode. In some embodiments of process 7, encryption algorithm \mathcal{E} is a stream cipher.

10.14 Some Complexity Analysis of Hidden Encrypted Data

Suppose that the encrypted data element $e_1 e_2 \dots e_{128}$ has 128 bits and these bits are hidden inside of 128 bits $r_1 r_2 \dots r_{128}$ of random noise. In an embodiment following process 7, block cipher Serpent is executed with cipher instructions 126 to encrypt the data element as $e_{162} \dots e_{128}$ before scatter map instructions 130 are applied to hide encrypted bits $e_1 e_2 \dots e_{128}$ in random noise $r_1 r_2 \dots r_{128}$ produced by random number generator 128.

The hiding of encrypted bits $e_1 e_2 \dots e_{128}$ by scatter map instructions 130 is shown in FIG. 11. Each encrypted data element is 16 bytes. In FIG. 11, the scatter size $n=256$ bits. For a scatter size of $n=256$, there are 256 possible locations to hide the first bit e_1 ; 255 locations to hide the second bit e_2 ; \dots ; and 129 locations to hide the 128th bit of encrypted data element e_{128} . Note that $|\mathcal{L}_{128,256}| > 10^{192}$. The complexity 10^{192} of finding the scatter map σ in $\mathcal{L}_{128,256}$ is substantially greater than the complexity of Serpent which is at most $2^{256} < 10^{77}$. In embodiments where more the noise size is larger, the complexity of finding the scatter map can be much greater than 10^{192} .

When Eve does not receive any auxiliary information (that is, rule 4 holds), it is extremely unlikely that Eve can extract any information about the bit locations even after Eve observes 625,000 encrypted data elements, each hidden in 128 bits of noise. If Eve has the computing power to brute-force search through each element $\sigma \in \mathcal{L}_{128,256}$ and subsequently to find data element $e_1 \dots e_{128}$, Eve still has no way of knowing if this particular σ is the one that Alice used to hide encrypted bits $e_1 e_2 \dots e_{128}$. Eve needs some auxiliary information.

10.15 the Scatter Map Process Hides One-Time Locks

Consider the following cryptographic method. Alice places her one-time lock a on message m and transmits $m \oplus a$ to Bob. Bob applies his one-time lock b and sends $m \oplus a \oplus b$ back to Alice. Alice removes her lock, by applying a to $m \oplus a \oplus b$ and sends $m \oplus b$ back to Bob. Bob removes lock b from $m \oplus b$ to read message m . This method of one-time locks is vulnerable if Eve can see the three transmissions $m \oplus a$, $m \oplus a \oplus b$ and $m \oplus b$ because Eve can compute $m = (m \oplus a) + (m \oplus a \oplus b) + (m \oplus b)$.

In an embodiment, process 8 protects these one-time locks by using two distinct and independent scatter maps π_A , π_B to hide each transmission inside a new generation of random noise. Independent means that any information given to Eve about π_B tells Eve nothing about π_A and vice versa. In terms of conditional probabilities, independence

45

means $P(\pi_A=(l_1 \dots l_K) \in \mathcal{L}_{K,n} | \pi_B=(j_1 \dots j_K) = P(\pi_A=(l_1 \dots l_K) \in \mathcal{L}_{K,n})$. Using these independent scatter maps, Eve is no longer able to see the three transmissions $m \oplus a$, $m \oplus a \oplus b$ and $m \oplus b$ because the encrypted data $m \oplus a$, and the twice encrypted data $m \oplus a \oplus b$ and the second party encrypted data $m \oplus b$ are each hidden inside of a new generation of random noise.

Hiding Process 8. Scattered One Time Locks

Alice and Bob possess secret scatter maps $\pi_A=(l_1, l_2 \dots l_K)$, $\pi_B=(j_1, j_2 \dots j_K) \in \mathcal{L}_{K,n}$.

Eve does not know K and π_A and π_B .

while Alice has more k -bit message blocks $m_1 m_2 \dots m_K$ to encrypt

{
Alice generates her lock $a_1 a_2 \dots a_K$ and noise $r_1 r_2 \dots r_{n-k}$ from her random noise generator.
Per definition 3, Alice uses π_A to set $s_{l_1}=a_1 \oplus m_1 \dots s_{l_K}=a_K \oplus m_K$.
Using noise $r_1 r_2 \dots r_K$ Alice completes scatter $S=(s_1 \dots s_n)$ and sends to Bob.
Bob extracts $a_1 \oplus m_1 \dots a_K \oplus m_K$ from S , using scatter map π_A .
Bob generates his lock $b_1 b_2 \dots b_K$ and noise $q_1 q_2 \dots q_{n-K}$ from his random noise generator.
Per definition 3, Bob uses π_B to set $t_{j_1}=b_1 \oplus a_1 \oplus m_1 \dots t_{j_K}=b_K \oplus a_K \oplus m_K$.
Using noise $q_1 q_2 \dots q_K$ Bob completes scatter $\mathcal{T}=(t_1 \dots t_n)$ and sends to Alice.
Alice extracts $b_1 \oplus a_1 \oplus m_1 \dots b_K \oplus a_K \oplus m_K$ from \mathcal{T} , using scatter map π_B .
Alice removes her lock $a_1 \dots a_K$ by computing $a_1 \oplus (b_1 \oplus a_1 \oplus m_1) \dots a_K \oplus (b_K \oplus a_K \oplus m_K)$. Alice generates noise $p_1 p_2 \dots p_{n-K}$ from her random noise generator.
Per definition 3, Alice uses π_A to set $u_{l_1}=b_1 \oplus m_1 \dots u_{l_K}=b_K \oplus m_K$.
Using noise $p_1 p_2 \dots p_K$ Alice completes scatter $\mathcal{U}=(u_1 \dots u_n)$ and sends to Bob.
Bob extracts $b_1 \oplus m_1 \dots b_K \oplus m_K$ from \mathcal{U} , using scatter map π_A .
Bob removes his lock $b_1 \dots b_K$ by computing $b_1 \oplus (b_1 \oplus m_1) \dots b_K \oplus (b_K \oplus m_K)$.
}

In an alternative embodiment, Alice and Bob use a third, distinct scatter map π_C , created independently from π_A and π_B . Scatter map π_C helps Alice scatter $b_1 \oplus m_1 \dots b_K \oplus m_K$ after removing her lock. This alternative embodiment is shown in process 9.

Hiding Process 9. Scattered One Time Locks with 3 Scatter Maps

Alice and Bob possess secret scatter maps $\pi_A=(l_1, l_2 \dots l_K)$, $\pi_B=(j_1, j_2 \dots j_K) \in \mathcal{L}_{K,n}$ and $\pi_C=(i_1, i_2 \dots i_K) \in \mathcal{L}_{K,n}$. Eve does not know π_A and π_B and π_C .

while Alice has more K -bit message blocks $m_1 m_2 \dots m_K$ to encrypt

{
Alice generates her lock $a_1 a_2 \dots a_K$ and noise $r_1 r_2 \dots r_{n-K}$ from her random noise generator.
Per definition 3, Alice uses π_A to set $s_{l_1}=a_1 \oplus m_1 \dots s_{l_K}=a_K \oplus m_K$.
Using noise $r_1 r_2 \dots r_K$ Alice completes scatter $S=(s_1 \dots s_n)$.
Alice sends $S=(s_1 \dots s_n)$ to Bob.
Bob extracts $a_1 \oplus m_1 \dots a_K \oplus m_K$ from S , using scatter map π_A .
Bob generates his lock $b_1 b_2 \dots b_K$ and noise $q_1 q_2 \dots q_{n-K}$ from his random noise generator.

46

Per definition 3, Bob uses π_B to set $t_{j_1}=b_1 \oplus a_1 \oplus m_1 \dots t_{j_K}=b_K \oplus a_K \oplus m_K$.

Using noise $q_1 q_2 \dots q_K$ Bob completes scatter $\mathcal{T}=(t_1 \dots t_n)$ and sends to Alice.

Alice extracts $b_1 \oplus a_1 \oplus m_1 \dots b_K \oplus a_K \oplus m_K$ from \mathcal{T} , using scatter map π_B .

Alice removes her lock $a_1 \dots a_K$ by computing $a_1 \oplus (b_1 \oplus a_1 \oplus m_1) \dots a_K \oplus (b_K \oplus a_K \oplus m_K)$. Alice generates noise $p_1 p_2 \dots p_{n-K}$ from her random noise generator.

Per definition 3, Alice uses π_C to set $u_{i_1}=b_1 \oplus m_1 \dots u_{i_K}=b_K \oplus m_K$.

Using noise $p_1 p_2 \dots p_K$ Alice completes scatter $\mathcal{U}=(u_1 \dots u_n)$ and sends to Bob.

Bob receives $\mathcal{U}=(u_1 \dots u_n)$.

Bob extracts $b_1 \oplus m_1 \dots b_K \oplus m_K$ from \mathcal{U} , using scatter map π_C .

Bob removes his lock $b_1 \dots b_K$ by computing by $b_1 \oplus (b_1 \oplus m_1) \dots b_K \oplus (b_K \oplus m_K)$.

In an embodiment of process 9, scatter maps π_A , π_B and π_C are executed by scatter map instructions 130 (FIG. 1B) and these instructions follow definition 3. In FIG. 2B, processor system 258 executes scatter map process instructions 130 during the execution of scatter maps π_A , π_B and π_C . In an embodiment, scatter map process instructions 130 for π_A , π_B and π_C are stored in memory system 256 (FIG. 2B).

In an embodiment of process 9, output system 252 in FIG. 2B is used during the step Alice sends $S=(s_1 \dots s_n)$ to Bob. Output system 252 is part of sending machine 102 in FIG. 1A. In an embodiment of process 9, input system 254 in FIG. 2B is used during the step Bob receives $\mathcal{U}=(u_1 \dots u_n)$. Input system 254 is a part of receiving machine 112 in FIG. 1A.

In other alternative, embodiments, the message size K is known to Eve.

In preferred embodiments, each scatter transmission should use a new lock and new noise. For example, if due to a failed transmission, Alice or Bob generated new noise but transmitted the same values of $a_1 \oplus m_1 \dots a_K \oplus m_K$ and $b_1 \oplus m_1 \dots b_K \oplus m_K$ and $b_1 \oplus a_K \oplus m_1 \dots b_K \oplus a_K \oplus m_K$, then Eve could run a matching or correlation algorithm between the scatters S , \mathcal{T} or \mathcal{U} in order to extract a permutation of message $m_1 \dots m_K$. During any kind of failed transmission, Alice and Bob should generate new locks from their respective random noise generators, just as they have to do for every iteration of the while loop in process 8.

In process 8, Alice's lock $a_1 \dots a_K$ is generated from her random noise generator. Hence, for every $(x_1, \dots, x_K) \in \{0, 1\}^K$, the probability $P(a_1 \oplus m_1 = x_1, \dots, a_K \oplus m_K = x_K) = 2^{-K}$. Similarly, Bob's lock $b_1 \dots b_K$ is generated from his random noise generator, so the probability $P(b_1 \oplus m_1 = x_1, \dots, b_K \oplus m_K = x_K) = 2^{-K}$ for every $(x_1, \dots, x_K) \in \{0, 1\}^K$.

10.16 Key Exchange

A Diffie-Hellman exchange is a key exchange method where two parties (Alice and Bob)—that have no prior knowledge of each other—jointly establish a shared secret over an unsecure communications channel. Sometimes the first party is called Alice and the second party is called Bob. Before the Diffie-Hellman key exchange is described it is helpful to review the mathematical definition of a group. A group G is a set with a binary operation $*$ such that the following four properties hold: (i.) The binary operation $*$ is closed on G . This means $a*b$ lies in G for all elements a and b in G . (ii.) The binary operation $*$ is associative on G . That is, $a*(b*c)=(a*b)*c$ for all elements a , b , and c in G (iii.)

47

There is a unique identity element e in G , where $a*e=e*a=a$.
(iv). Each element a in G has a unique inverse denoted as a^{-1} . This means $a*a^{-1}=a^{-1}*a=e$.

$g*g$ is denoted as g^2 ; $g*g*g*g*g$ is denoted as g^5 .
Sometimes, the binary operation $*$ will be omitted so that $a*b$ is expressed as ab .

The integers $\{\dots, -2, -1, 0, 1, 2, \dots\}$ with respect to the binary operation $+$ are an example of an infinite group. 0 is the identity element. For example, the inverse of 5 is -5 and the inverse of -107 is 107 .

The set of permutations on n elements $\{1, 2, \dots, n\}$, denoted as S_n , is an example of a finite group with $n!$ elements where the binary operation is function composition. Each element of S_n is a function $p: \{1, 2, \dots, n\} \rightarrow \{1, 2, \dots, n\}$ that is 1 to 1 and onto. In this context, p is called a permutation. The identity permutation e is the identity element in S_n , where $e(k)=k$ for each k in $\{1, 2, \dots, n\}$.

If H is a non-empty subset of a group G and H is a group with respect to the binary group operation $*$ of G , then H is called a subgroup of G . H is a proper subgroup of G if H is not equal to G (i.e., H is a proper subset of G). G is a cyclic group if G has no proper subgroups.

Define $A_n = \mathbb{Z}_n - [0] = \{[1], \dots, [n-1]\}$; in other words, A_n is the integers modulo n with equivalence class $[0]$ removed. If $n=5$, $[4]*[4]=[16 \bmod 5]=[1]$ in $(\mathbb{Z}_5, *)$. Similarly, $[3]*[4]=[12 \bmod 5]=[2]$ in $(\mathbb{Z}_5, *)$. Let (a, n) represent the greatest common divisor of a and n . Let $U_n = \{[a] \in A_n : (a, n)=1\}$. Define binary operator on U_n as $[a]*[b]=[ab]$, where ab is the multiplication of positive integers a and b . Then $(U_n, *)$ is a finite, commutative group.

Suppose g lies in group $(G, *)$. This multiplicative notation works as follows: $g^2=g*g$. Also $g^3=g*g*g$; and so on. This multiplicative notation (superscripts) is used in the description of the Diffie-Hellman key exchange protocol described below.

For elliptic curves the Weierstrauss curve group operation geometrically takes two points, draws a line through these two points, finds a new intersection point and then reflects this new intersection point about the y axis. When the two points are the same point, the commutative group operation computes a tangent line and then finds a new intersection point.

In another embodiment, elliptic curve computations are performed on an Edwards curve over a finite field. When the field K does not have characteristic two, an Edwards curve is of the form: $x^2+y^2=1+dx^2y^2$, where d is an element of the field K not equal to 0 and not equal to 1 . For an Edwards curve of this form, the group binary operator $*$ is defined as

$$(x_1, y_1) * (x_2, y_2) = \left(\frac{x_1 y_2 + x_2 y_1}{1 + dx_1 x_2 y_1 y_2}, \frac{y_1 y_2 - x_1 x_2}{1 - dx_1 x_2 y_1 y_2} \right),$$

where the elements of the group are the points (x_1, y_1) and (x_2, y_2) . The definition of $*$ defines elliptic curve computations that form a commutative group. For more information on Edwards curves, refer to the math journal paper [57].

In an alternative embodiment, elliptic curve computations are performed on a Montgomery curve over a finite field. Let K be the finite field over which the elliptic curve is defined. A Montgomery curve is of the form $By^2=x^3+Ax^2+x$, for some field elements A, B chosen from K where $B(A^2-4) \neq 0$. For more information on Montgomery curves, refer to the publication [58].

There are an infinite number of finite groups and an infinite number of these groups are huge. The notion of huge

48

means the following: if 2^{1024} is considered to be a huge number based on the computing power of current computers, then there are still an infinite number of finite, commutative groups with each group containing more than 2^{1024} elements.

Before the Diffie-Hellman key exchange is started, in some embodiments, Alice and Bob agree on a huge, finite commutative group $(G, *)$ with group operation $*$ and generating element g in G , where g has a huge order. In some embodiments, Alice and Bob sometimes agree on group $(G, *)$ and element g before the key exchange starts; g is assumed to be known by Eve. The group operations of G are expressed multiplicatively as explained previously.

In a standard Diffie-Hellman key exchange, Alice executes steps 1 and 3 and Bob executes steps 2 and 4.

1. Alice randomly generates private key a , where a is a large natural number, and sends g^a to Bob.
2. Bob randomly generates private key b , where b is a large natural number, and sends g^b to Alice.
3. Alice computes $(g^b)^a$.
4. Bob computes $(g^a)^b$.

After the key exchange is completed, Alice and Bob are now in possession of the same shared secret g^{ab} . The values of $(g^b)^a$ and $(g^a)^b$ are the same because G is a commutative group. Commutative means $ab=ba$ for any elements a, b in G .

10.17 Hiding a Public Key Exchange

The Diffie-Hellman exchange [54, 55] is vulnerable to active man-in-the-middle attacks [59, 60, 61]. To address man-in-the-middle attacks, processes 10 and 11 show how to hide public session keys during a key exchange. In some embodiments, Alice and Bob have previously established secret scatter map $\sigma=(l_1, l_2, \dots, l_m)$ and authentication key π with a one-time pad [35]. In another embodiment, Alice and Bob may establish σ and K with a prior (distinct) Diffie-Hellman exchange that is resistant to quantum computers, executing Shor's algorithm or an analogue of Shor's algorithm [63]. Alternatively, Alice and Bob may establish σ and π via a different channel.

Let h_k denote an MAC (e.g., HMAC or [65]) function which will be used to authenticate the scattered transmission. The use of h_k helps hinder the following attack by Eve. An active Eve could flip a bit at bit location l in the scattered transmission. If no authentication occurs on the noise and the hidden key bits, then upon Alice resending a scattered transmission due to Alice and Bob not arriving at the same session key secret, Eve gains information that l lies in σ . If the scattered transmission S is not authenticated, Eve's manipulation of the bits in S helps her violate rule 4.

Hiding Process 10. First Party Hiding and Sending a Public Key to a Second Party

Alice's random noise generator generates and computes private key a .

Alice uses group operation $*$ to compute public key $g^a=k^1 k^2 \dots k_m$ from private key a and generator g .

Alice generates noise $r_1 r_2 \dots r_p$ from her random noise generator.

Per definition 3, Alice uses σ to find the hiding locations and set $s_{l_i}=k_{l_i} \dots s_{l_m}=k_m$.

Alice stores noise $r_1 r_2 \dots$ in the remaining noise locations, resulting in $S=(s_1 \dots s_n)$.

Alice computes $h_k(S)$.

Alice sends S and $h_k(S)$ to Bob.

Bob receives S' and $h_k(S)$ from Alice.

Bob computes $h_k(S')$ and checks it against $h_k(S)$.

49

If $h_k(S')$ is valid
 {
 Bob uses σ to extract $g^a = k_1 \dots k_m$ from S.
 Bob computes shared secret g^{ab} .
 }
 else
 {
 Bob rejects S' and asks Alice to resend S.
 }

Note that Alice sends S and Bob receives S' because during the transmission from Alice to Bob S may be tampered with by Eve or S may change due to physical effects. In an embodiment of process 10, Bob's steps are performed in receiving machine 112. In an embodiment of process 10, Alice's steps are performed in sending machine 102. In an embodiment of process 10, private key(s) 103 is a and public key(s) 104 is g^a . In an embodiment of process 10, scatter map σ finds the hiding locations with scatter map instructions 130. In an embodiment, the size of the transmission S (hidden public keys 109) is $n=8192$ bits and the noise size $p=6400$. According to $\sigma=(l_1, l_2 \dots l_m)$, the k th bit of P is stored in bit location l_k . Generator g is an element of a commutative group $(G, *)$ with a huge order. In some embodiments, G is a cyclic group and the number of elements in G is a prime number. In an embodiment, generator g has an order $o(g) > 10^{80}$. In another embodiment, generator g has an order $o(g)$ greater than 10^{1000} . In an embodiment, Alice randomly generates with non-deterministic generator 942 in FIG. 9A, which is an instance of random generator 128 and computes private key a with private key instructions 124. In an embodiment, Alice's public key instructions 126 compute her public key as $g^a = g * \dots * g$ where g is multiplied by itself a times, using the group operations in $(G, *)$. In some embodiments, the private key is randomly selected from the positive integers $\{1, 2, 3, \dots, o(g)-1\}$.

Hiding Process 11. Second Party Hiding and Sending a Public Key to the First Party

Bob's random noise generator generates and computes private key b .
 Bob uses group operation $*$ to compute public key $g^b = j_1 j_2 \dots j_m$ from Bob's private key b and generator g .
 Bob generates noise $q_1 q_2 \dots q_p$ from his random noise generator.
 Per definition 3, Bob uses σ to find the hiding locations and set $s_{l_1} = j_1 \dots s_{l_m} = j_m$.
 Bob stores noise $q_1 q_2 \dots q_p$ in the remaining noise locations, resulting in $\mathcal{T} = (t_1 \dots t_n)$.
 Bob computes $h_k(\mathcal{T})$.
 Bob sends \mathcal{T} and $h_k(\mathcal{T})$ to Alice.
 Alice receives \mathcal{T}' and $h_k(\mathcal{T})$ from Bob.
 Alice computes $h_k(\mathcal{T}')$ and checks it against $h_k(\mathcal{T})$.
 If $h_k(\mathcal{T}')$ is valid
 {
 Alice uses σ to extract $g^b = j_1 \dots j_m$ from \mathcal{T} .
 Alice computes shared secret g^{ba} .
 }
 else
 {
 Alice rejects \mathcal{T}' and asks Bob to resend \mathcal{T} .
 }

Note that Bob sends \mathcal{T} and Alice receives \mathcal{T}' because during the transmission from Bob to Alice \mathcal{T} may be tampered with by Eve or \mathcal{T} may change due to physical effects. In an embodiment of process 11, Alice's steps are performed in receiving machine 112. In an embodiment of process 11, Bob's steps are performed in sending machine

50

102. In an embodiment of process 11, private key(s) 103 is b and public key(s) 104 is g^b . In an embodiment of process 11, scatter map σ finds the hiding locations with scatter map instructions 130.

5 In an embodiment, Bob randomly generates with non-deterministic generator 952 in FIG. 9B, which is an instance of random generator 128 and computes private key b with private key instructions 124. In an embodiment, Bob's public key instructions 126 compute his public key as $g^b = g * \dots * g$ where g is multiplied by itself b times, using the group operations in $(G, *)$. In some embodiments, the private key b is randomly selected from the positive integers $\{1, 2, 3, \dots, o(g)-1\}$. In some embodiments of 10 and 11, the public keys are computed with elliptic curve computations over a finite field; in other words, G is an elliptic curve group. In other embodiments, the public keys are RSA public keys. In some embodiments, the public keys are public session keys, which means the public session keys change after every transmission S in process 10 and after every transmission \mathcal{T} in process 11.

In some embodiments, hiding a public key during an exchange between Alice and Bob has an advantage over hiding a symmetric key: processes 10 and 11 can be used by Alice and Bob, before a subsequent encrypted communication, to communicate a short authentication secret (SAS) via a different channel.

Let a, b be Alice and Bob's private keys, respectively. Let e_1, e_2 be Eve's private keys. For a key exchange, if Eve is in the middle, Eve computes $g^{e_1 a}$ with Alice; Eve computes $g^{e_2 b}$ with Bob. When Alice and Bob verify their SAS with high probability $g^{e_1 a} \neq g^{e_2 b}$ when $|G|$ is huge. Thus, $h_k(g^{e_1 a}) \neq h_k(g^{e_2 b})$ with high probability, regardless of whether Eve's private keys satisfy $e_1 \neq e_2$. By communicating their short authentication secret to each other via a different channel, Alice and Bob can detect that Eve captured σ before processes 10 and 11 were executed. Eve cannot duplicate the SAS secret because Eve doesn't know Alice's private key a and Eve doesn't know Bob's private key b . This type out-of-channel authentication won't work for symmetric keys hidden inside noise. Furthermore, one anticipates that Eve will try to capture σ since complexity analysis can show that if Eve doesn't know σ , the complexity for Eve performing a man-in-the-middle can be substantially greater than the conjectured complexity of the public session keys when the noise size is sufficiently large.

It is important to recognize the difference between SAS and hiding the public keys in random noise: they are complementary methods. SAS helps notify Alice and Bob that a man-in-the-middle on a standard Diffie-Hellman exchange has occurred, but SAS DOES NOT stop a man-in-the-middle attack. SAS does not stop an adversary who has unforeseen computing power or unknown mathematical techniques. The standard Diffie-Hellman exchange depends upon the conjectured computational complexity of the underlying commutative group operation $*$ on G . If Eve is recording all network traffic, hiding public session keys inside random noise can stop Eve from breaking the standard key exchange even if Eve has already discovered a huge, computational or mathematical breakthrough on the underlying group G or if Eve finds one at some point in the future. Public keys that are resistant to quantum computing algorithms such as Shor's algorithm are quite large (e.g., 1 million bytes and in some cases substantially larger than 1 million bytes). In contrast, 1024 bytes of hidden public keys inside noise can provide adequate protection against quantum algorithms; in other embodiments, 4096 bytes of hidden public keys inside noise provides strong protection against

51

quantum algorithms. Processes 10 and 11 complementary property to SAS depends upon Eve not obtaining σ ; in some embodiments, a one-time pad may be feasible to establish σ between Alice and Bob.

11 STRUCTURAL STABILITY

Structural stability is a mathematical and engineering tool that we want to apply to computer programs because a register machine program can be modeled as a discrete, autonomous dynamical system. When a perturbed instruction is close enough to an original instruction that is stable, then the computational behavior of the program will not change. For this reason, structural stability can be used as a mathematical tool to help design a solution that hinders malware sabotage. Overall, we briefly describe topological spaces, metric spaces and structural stability before we specify how to build a stable sequential machine.

11.1 Topological Spaces

A topology [23] on a set X is a collection \mathcal{T} of subsets of X having the following properties: (a) \emptyset and X are both in \mathcal{T} ; (b) The union of the elements of any subcollection of \mathcal{T} is in \mathcal{T} ; (c) The intersection of the elements of any finite subcollection of \mathcal{T} is in \mathcal{T} . A set X for which a topology \mathcal{T} has been specified is called a topological space. A subset U of X is called open in this topology if U belongs to the collection \mathcal{T} .

Let \mathbb{R} be the real numbers. For real numbers a and b , the open intervals $(a, b) = \{x \in \mathbb{R} : a < x < b\}$ form a basis for the standard topology on \mathbb{R} , generated from arbitrary unions of open intervals and finite intersections of open intervals.

Let X and Y be topological spaces. The function $f: X \rightarrow Y$ is continuous if for any open subset U of Y , the inverse image $f^{-1}(U) = \{x \in X : f(x) \text{ lies in } U\}$ is open in X 's topology. The function $h: X \rightarrow Y$ is a homeomorphism if h is continuous, h is one-to-one and onto, and h 's inverse $h^{-1}: Y \rightarrow X$ is continuous.

11.2 Topological Conjugacy

A discrete, dynamical system is a function $f: X \rightarrow X$, where X is a topological space. The orbit of p is $\{f^n(p) : n \in \mathbb{N}\}$, which is the set of points, obtained by iterating f on initial point p . Consider discrete, dynamical systems $f: X \rightarrow X$ and $g: Y \rightarrow Y$. f and g are topologically conjugate if f and g are continuous functions and there exists a homeomorphism $h: X \rightarrow Y$ such that $h \circ f = g \circ h$. Topologically conjugate functions exhibit equivalent dynamics. For example, if f is topologically conjugate to g via h and p is a fixed point for f , then $h(p)$ is a fixed point for g .

Topological conjugacy is a useful notion for computation because after a register machine has halted, its halted machine configuration represents what the machine has computed. Furthermore, each halted machine configuration corresponds to a fixed point (halting point) of a dynamical system that faithfully models the register machine. If h is a topological conjugacy with $h \circ f = g \circ h$, then p is a fixed point of f if and only if $h(p)$ is a fixed point. Hence, a topological conjugacy between machines M_1 and M_2 induces a one-to-one correspondence between the halting configurations of M_1 and M_2 .

11.3 Metric Spaces

A metric space is a set X and a function (metric) $d: X \times X \rightarrow \mathbb{R}$ (real numbers) such that the following three conditions hold.

52

1. $d(a, b) \geq 0$ for all $a, b \in X$ where equality holds if and only if $a = b$.
2. $d(a, b) = d(b, a)$ for all $a, b \in X$. (Symmetric).
3. $d(a, b) \leq d(a, c) + d(c, b)$ for all $a, b, c \in X$. (Triangle inequality).

For $\epsilon > 0$, define the ϵ -ball $B_d(x, \epsilon) = \{y \in X : d(x, y) < \epsilon\}$. The collection of all ϵ -balls, where $\epsilon > 0$ and $x \in X$, is a basis for a metric topology on X , induced by metric d .

11.4 C^0 Structural Stability

Let (X, d) be a metric space. The C^0 distance between functions $f: X \rightarrow X$ and $g: X \rightarrow X$ is given by $\rho_0(f, g) = \sup\{d(f(x), g(x)) : x \in X\}$, where \sup is the least upper bound. A function $f: X \rightarrow X$ is said to be C^0 structurally stable on X if there exists $\epsilon > 0$ such that whenever $\rho_0(f, g) < \epsilon$ for $g: X \rightarrow X$, then f is topologically conjugate to g . In other words, a dynamical system f is structurally stable if for all dynamical systems g that are close to f , then f is topologically conjugate to g .

12 A STRUCTURALLY STABLE MACHINE

In our invention(s), we have two design criteria, developed in section 7:

- A. Build instructions that are invariant to small changes in their representation.
- B. Hinder the adversary from figuring out how to manipulate these machine instructions.

We start with a Turing complete, register machine, as a starting point for building computation that satisfies our two design criteria for our invention(s). We describe machine procedures that transform these register machine instructions: these transformation procedures, implemented in hardware, execute the functionality of the register machine instructions so that the computation is stable under small changes to the transformed instructions. In an embodiment, these procedures execute in Protected Machine Hardware **1502** of FIG. **15** (blue region). Furthermore, these transformed instructions also satisfy design criterion B.

12.1 A Register Machine

Below is a description of instructions for an embodiment of a register machine, executed by processor system **258** in FIG. **2B**. In other embodiments, similar register machine instructions comprise a hardware computing machine. The transformation procedures (machine procedures) are provided in subsections 12.2, 13.1 and 13.3. In some embodiments, these procedures are implemented in Protected Machine Hardware **1502** in FIG. **15**.

SET A **14** stores **14** in register A.

ADD B C adds the contents of registers B and C and stores the sum in register B.

SUB C 13 subtracts 13 from the contents of register C and stores the difference in register C.

MUL D C multiplies the contents of register D and register C and stores the product in register D.

DIV A 7 divides the contents of register A by 7 and stores the quotient in register A. If A contains 26, then after DIV A 7 is executed register A contains 3.

MOD A 7 divides the contents of register A by 7 and stores the remainder in A. If register A contains 26, then after MOD A 7 is executed register A contains 5.

JMP L_AB updates the program counter to execute the instruction tagged by label L_AB. If L_AB contains 37,

53

then instruction 37 will be executed next. JMP acts as an unconditional branch instruction.

IF A B executes the next instruction if the contents of register A equals the contents of register B. Otherwise, the next instruction is skipped. IF is an embodiment of a conditional branch instruction.

IFN C 5 executes the next instruction if the contents of register C are not equal to 5. If register C contains 5, the next instruction is skipped. IFN is an embodiment of a conditional branch instruction.

IFGT C 12 executes the next instruction if the contents of register C are greater than 12. If register C contains a number less than 13, the next instruction is skipped. IFGT is an embodiment of a conditional branch instruction.

The opcode for instruction name SET represents SET in terms of bits. The other instruction names {ADD, SUB, MUL, DIV, MOD, JMP, IF, IFN, IFGT, STORE, GET} each have their own unique opcode in terms of bits. An opcode is mathematically defined as a function $\mathcal{O}: \{\text{SET, ADD, SUB, MUL, DIV, MOD, JMP, IF, IFN, IFGT, STORE, GET}\} \rightarrow \{0, 1\}^n$ such that \mathcal{O} is 1-to-1. One-to-one means \mathcal{O} maps two different instruction names to two distinct n-bit strings. A valid instruction starts with an instruction name, followed by one or two operands. In instruction IFGT C 12, register C is the first operand and the number 12 is the second operand. The JMP instruction is the only instruction with one operand.

Program 1 implements a greatest common divisor algorithm with these register machine instructions. In section 12.2, stable machine (SM) tools are described that transform these instructions so that the program instructions are hidden, stable and unpredictable. Any symbols on the same line of program 1, that follow a semicolon, are comments.

Program 1. Greatest Common Divisor

```
SET A 105 ; Instruction 0
SET B 21 ; Instruction 1
; IF (A==0) execute instruction 3.
IF A 0 ; Label: IFAEQ0. Instruction 2
JMP SET_AB ; Instruction 3. Jump to label SET_AB
; IF (B==0) execute instruction 5.
IF 0 ; Instruction 4.
JMP SET_BA ; Instruction 5. Jump to label SET_BA
; IF (A>B) execute instruction 7.
IFGT A B ; Instruction 6.
SUB A B ; Instruction 7. Store A-B in reg A
; IF (B>A) execute instruction 9.
IFGT B A ; Instruction 8.
SUB B A ; Instruction 9. Stores B-A in reg B
; IF (A !=B) execute instruction 11.
IFN A B ; Instruction 10.
JMP IFAEQ0 ; Instruction 11. Jump to label IFAEQ0
SET A B ; Label: SET_AB. Instruction 12
SET B A ; Label: SET_BA. Instruction 13
```

12.2 Randomizing Instruction Opcodes

We describe a procedure that randomizes opcodes such that each opcode is a minimal Hamming distance apart. Our random opcode procedure is a computational tool for helping us achieve our two design criteria.

First, we review some definitions from information theory. $\{0, 1\}^n$ is the collection of all n-bit strings, where each binary string a in $\{0, 1\}^n$ can represent an opcode of a virtual machine instruction. Sometimes b in $\{0, 1\}^p$ can represent an operand of a virtual machine instruction, and in some cases $n \neq p$.

54

Let $a=a_1, \dots, a_n$ and $b=b_1, \dots, b_n$ be binary strings of length n. For each n, the Hamming distance [67] (Hamming metric) is defined as

$$d(a, b) = \sum_{i=1}^n |a_i - b_i|.$$

It is easy to verify that $(\{0, 1\}^n, d)$ is a metric space per section 11.3.

Note that the Hamming distance between string 0010 and string 0111 is expressed as $d(0010, 0111)=2$. Consider string $c=c_1 \dots c_n$ in $\{0, 1\}^n$. A Hamming ball $H(c, m)=\{a \in \{0, 1\}^n: d(c, a) \leq m\}$ has center c and radius m.

q is a non-deterministic generator, as shown in non-deterministic generator 942 of FIG. 9A, non-deterministic generator 952 of FIG. 9B, and non-deterministic generator 1504 of FIG. 15. In some embodiments, q measures quantum events and returns one or more bits, generated from photons in LED 946 of FIG. 9A; or photons in LED 956 of FIG. 9B; or photons from LED 1002 of FIG. 10; and further described in subsection 10.5. (The emission and absorption of photons are quantum events.) (Also, see subsection 10.5, FIG. 1B, FIG. 1D, FIG. 9A, FIG. 9B, FIG. 9C, FIG. 10 and FIG. 15.) When q is executed, it returns a 0 or a 1, based on a measurement of a non-deterministic process. In some embodiments, the measurement is a quantum measurement (the emission of photons in 1002 of FIG. 10, and the absorption of one or more photons with energy $h\nu$ by phototransistor 944 in FIG. 9A, or the absorption of one or more photons with energy $h\nu$ by photodiode 954 in FIG. 9B.). In procedures 1, 2, and 3, we use q to help construct m distinct random opcodes each of length n that are pairwise a minimum Hamming distance of $2l+1$ bits apart. These procedures build random opcodes that are stable, when there are at most l bits of sabotage on a single opcode. In other words, a random opcode I_j is the center of a Hamming ball with radius l. In terms of geometry, all opcodes in $H(I_j, l)$ can still be resolved to the correct opcode I_j .

Machine procedure 1 builds an n-bit random opcode $I_{j,1} I_{j,2} \dots I_{j,n}$ used by instruction I_j , after random bit generator q measures n random bits. In an embodiment, machine procedure 1 uses a non-deterministic generator q as shown in non-deterministic generator 942 of FIG. 9A, or non-deterministic generator 952 of FIG. 9B, or non-deterministic generator 1504 of FIG. 15. In some embodiments, q measures quantum events and returns one or more bits, generated from photons in LED 946 of FIG. 9A; or photons in LED 956 of FIG. 9B; or photons from LED 1002 of FIG. 10; and further described in subsection 10.5.

Machine Procedure 1. Random Opcode

```
Input: n
set k:=1
while k≤n
{
  set  $I_{j,k}$  to a random bit measured by q
  increment k by 1
}
Output:  $I_{j,1} I_{j,2} \dots I_{j,n}$ 
```

The jth random opcode, called I_j , is n bits long. Machine procedure 2 begins with m random opcodes I_1, I_2, \dots, I_m as input. Machine procedure 2 finds two distinct opcodes in $\{I_1, I_2, \dots, I_m\}$ that are a smallest Hamming distance h_u apart.

55

Machine Procedure 2. Minimal Hamming Pair

Input: m, n, and opcodes $\{I_1, I_2, \dots, I_m\}$ set $h_\mu = n+1$ set $i=1$ while $i < m$

{

set $j=i+1$ while $j \leq m$

{

set $h = d(I_i, I_j)$ in $\{0,1\}^n$ if ($h < h_\mu$)

{

set $h_\mu = h$ set $i_\mu = i$ set $j_\mu = j$

}

increment j by 1

}

increment i by 1

}

Output: $I_{i_\mu}, I_{j_\mu}, h_\mu$

Machine procedure 3 uses machine procedures 1 and 2 to build m random opcodes (n-bit codes) that are pairwise at least a Hamming distance of $2l+1$ apart.

Machine Procedure 3. Minimal Hamming Distance

Input: l, m, n

call procedure 1 m times with input n. output is I_1, \dots, I_m call procedure 2 with input I_1, \dots, I_m . output is $I_{i_\mu}, I_{k_\mu}, I_{h_\mu}$ set $r:=0$ while $h_\mu < (2l+1)$

{

set $b:=2l+1-h_\mu$

do b times

{

use q to randomly choose positive integer k in the set $\{1, 2, \dots, n\}$ flip bit k in I_{i_μ}

}

execute procedure 2 with input I_1, \dots, I_m

increment r by 1

}

Output: I_1, I_2, \dots, I_n where $d(I_j, I_k) \geq 2l+1$ when $j \neq k$

Flip bit k means: if the kth bit is 0, then set the kth bit to 1; and if the kth bit is 1, then set the kth bit to 0. Variable r counts the number of repairs on a random opcode until all random opcodes are at least a Hamming distance of $2l+1$ -bits apart. $2l+1$ should be about 1 to 2.5 standard deviations less than $2/n$ so that the outer loop while $h_\mu < (2l+1)$ exits in a reasonable amount of time. ($n/2$ is the expected Hamming distance between two random n-bit codes, where each bit (0 or 1) occurs with probability $1/2$.)

If l is too large (e.g., $(2l+1) > n$), then the outer loop never exits and $r \rightarrow \infty$. To avoid long computing times, a variation of procedure 3 adds, before the outer loop, instruction

$$\text{set } l = \left\lfloor \frac{1}{4}(n - c\sqrt{n}) - \frac{1}{2} \right\rfloor,$$

where

$$1 \leq c \leq \frac{5}{2}.$$

56

(The standard deviation of uniformly random n-bit codes is $\sqrt{n}/2$.)

In our cryptographic model, Alice's m random opcodes I_1, I_2, \dots, I_m "act as her private keys." Hence, her random opcodes must be generated and stored in Protected Machine Hardware (blue region in FIG. 15) so that procedure 3 can help assure anonymity and valid execution of her instructions. It is also good practice for Alice to keep l private.

10 13 INSTRUCTION AND PROGRAM STABILITY

This section defines and specifies instruction and opcode stability, used in our hardware embodiments of machine procedures, 1, 2, 3, 4, 5, 6, and 7, and implemented in FIG.

15 15.

Definition 8. A set of opcodes (or operands) $\{I_1, \dots, I_m\}$ is s-bit stable if $\min\{d(I_j, I_k) : j \neq k\} \geq s$. In other words, if I_j and I_k are the closest opcodes (operands) in $\{I_1, \dots, I_m\}$, then $d(I_j, I_k) \geq s$.

Definition 9. An n-bit instruction I is s-bit stable if its opcode and operands are both s-bit stable.

Remark 4. The random opcodes generated by a successful exit of procedure 3 are $2l+1$ -bit stable.

Machine procedure 2 returns h_μ . h_μ is the minimum distance between any two distinct opcodes in $\{I_1, \dots, I_m\}$. A successful exit of machine procedure 3 means the loop while $h_\mu < (2l+1)$ exited; hence, $h_\mu \geq 2l+1$.

Using definitions 8, 9 and remark 4, we can explain the stability of programs, transformed by machine procedures 1, 2 and 3, in terms of the description in section 11. First, we need to define a metric on a space of programs, where each program is a finite sequence of transformed (procedures 1, 2 and 3) instructions (before hiding). Consider a program of transformed instructions $\mathcal{P}_1 = (I_1, I_2, \dots, I_m)$ and another program of transformed instructions $\mathcal{P}_2 = (J_1, J_2, \dots, J_m)$. If two programs have the same length m, define the distance between them as $\mathcal{D}(\mathcal{P}_1, \mathcal{P}_2) = \max\{d(I_k, J_k) : 1 \leq k \leq m\}$. If programs \mathcal{P} and \mathcal{Q} have different lengths, then define $\mathcal{D}(\mathcal{P}, \mathcal{Q}) = n$, where all program instructions in \mathcal{P} and \mathcal{Q} lie in $\{0, 1\}^n$.

In program $\mathcal{P}_1 = (I_1, I_2, \dots, I_m)$, for each k such that $1 \leq k \leq m$, machine procedure 3 can build each instruction I_k so that I_k is $2l+1$ -bit stable. Consider \mathcal{P}_1 as a dynamical system. Program \mathcal{P}_1 is structurally stable because whenever $\mathcal{D}(\mathcal{P}_1, \mathcal{P}_2) \leq l$ each instruction J_k in \mathcal{P}_2 that corresponds to I_k is resolved to the same opcode and same operand(s). Hence, the dynamical (computing) behavior of \mathcal{P}_1 is the same as program \mathcal{P}_2 : this means programs \mathcal{P}_1 and \mathcal{P}_2 perform the same task. Structural stability helps protect the integrity of the computation.

50 13.1 Hiding Operands and Opcodes in Noise

In 12.2, we described machine procedures 1, 2, and 3 for building random opcodes stable to small changes. When machine procedures 1, 2, and 3 are also applied to operands, the representation of the operands can become stable to small changes. In our model, Eve is a sentient adversary, so structural stability from classical mathematics alone does not provide enough mathematical firepower to build malware resistant computation. Hence, the purpose of design criteria B is to make a representation of each instruction that is computationally intractable for Eve to understand its meaning.

Machine procedure 4 builds a random permutation.

Machine Procedure 4. Random Permutation p

Input: n

65 set $p(1):=1$ set $p(2):=2 \dots$ set $p(n):=n$ set $k:=n$ while $k \geq 2$

57

```

{
  use q to randomly choose positive integer r in the set
    {1, . . . , k}
  set t:=p(r)
  set p(r):=p(k)
  set p(k):=t
  decrement k by 1
}
Output: Permutation p on {1, . . . , n}
Machine procedure 5 uses machine procedure 4 to build
a random substitution box.

```

Machine Procedure 5. Random Substitution Box σ

```

Input:  $\eta$ 
call machine procedure 4 with input  $n=2^\eta$  to create
  random permutation  $\sigma$ 
Output: Substitution Box  $\sigma$ , where  $\sigma$  maps an  $\eta$ -bit input
  to an  $\eta$ -bit output.

```

The rest of this subsection describes how to hide the meaning of the opcodes and operands from Eve. Machine procedures 4 and 5 have different purposes even though they both generate a random permutation. Machine procedure 5 constructs a σ that has size 2^η . In some embodiments, $\eta=8$ because 8 bits is a byte, and $n=16$ is more expensive to build in hardware. If $\eta=16$, then σ has size $2^{16}=65536$. Our goal is to store σ in protected machine hardware (FIG. 15) that is not accessible to Eve. If the operands in the base register machine have size equal to 64 bits before hiding in 64 bits of noise, then machine procedure 5 is called sixteen times to generate $\sigma_1 \sigma_2 \dots \sigma_{16}$ for the first operand. For the second operand, machine procedure 5 is called sixteen more times to generate $\sigma_{17} \dots \sigma_{32}$. In general, σ_i and σ_j are statistically independent when $i \neq j$, as result of using machine procedure 5.

Machine procedure 4 builds p to locate the 64 bits of the signal $b_1 b_2 \dots b_{64}$ inside the random noise. p lies in the symmetric group on $\{1, 2, \dots, n\}$, and determines where each bit of the signal (i.e., opcode or operand) is located: the i th bit b_i is stored at bit location $p(i)$, where $1 \leq i \leq 64$. When there are 64 bits of signal from the operand or opcode and 64 bits of quantum random noise, then the size of p is 128, i.e., $n=128$.

Random noise is measured and stored in the remaining 64 bit locations. The result is 128 bits of noise and signal, named $s_1 \dots s_{128}$. Subsequently, the 16 randomly generated sboxes σ_i with $1 \leq i \leq 16$ are applied to $s_1 s_2 \dots s_{128}$ as follows: $\sigma_1(s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8), \sigma_2(s_9 \dots s_{16}), \dots, \sigma_{16}(s_{121} s_{122} s_{123} s_{124} s_{125} s_{126} s_{127} s_{128})$, which is named $c_1 c_2 \dots c_{128}$. Next, a distinct random permutation τ is generated on $\{1, 2, \dots, 128\}$. τ is applied to $c_1 c_2 \dots c_{128}$, resulting in $c_{\tau(1)} c_{\tau(2)} \dots c_{\tau(128)}$. Then sixteen distinct sboxes $\alpha_1 \dots \alpha_{16}$ are randomly generated and applied to $c_{\tau(1)} c_{\tau(2)} \dots c_{\tau(128)}$ as follows: $\alpha_1(c_{\tau(1)} c_{\tau(2)} \dots c_{\tau(8)}) \dots \alpha_{16}(c_{\tau(121)} \dots c_{\tau(128)})$, resulting in $o_1 \dots o_{128}$.

In subsection 14, a birthday paradox statistical test is described in order to address potential attacks that involve stable instructions of size 128 bits. The birthday attack test performs 2^7 compilations of the same unmasked instruction. In 14, we also describe a statistical test to address multiple transformations via machine procedure 6 of the same instruction at different locations in a single SM compilation of the program. For each time the SM compilation tool is executed, all $p, \sigma_i, \tau, \alpha_i$ and randomized opcodes are statistically independent from each other and from all previous compilations. Also, within one SM compilation, the random noise ($n-k$ bits per opcode) generated for two identical opcodes at different locations in the program is statistically independent.

58

From our prior description, machine procedure 6 formally specifies hiding a k -bit opcode (or operand) $b_1 \dots b_k$ in $n-k$ bits of random noise. p and τ are distinct random permutations on $\{1, 2, \dots, n\}$. p determines the bit locations of $b_1 \dots b_k$ hidden inside of noise. $\sigma_1 \dots \sigma_p$ and $\alpha_1 \dots \alpha_p$ are random sboxes. In some embodiments, machine procedure 6 executes in protected machine hardware 1502 in FIG. 15. In some embodiments, the hiding of opcode bits $b_1 \dots b_k$ at bit locations $l_{p(1)} \dots l_{p(k)}$ is executed in hiding machine 190 in FIG. 1F.

Machine Procedure 6. Hide in Noise

```

Input:  $k, n$  and  $k$ -bit string  $b_1 b_2 \dots b_k$ 
call procedure 4 with input  $n$  to build random permutation
   $p$  on  $\{1, 2, \dots, n\}$ 
call procedure 1 with input  $n$  and store noise in every bit
  location  $l_1 l_2 \dots l_n$ 
set  $j:=1$ 
while  $j \leq k$ 
{
  set bit location  $l_{p(j)}:=b_j$ 
  increment  $j$  by 1
}
call procedure 5  $p$  times on input  $y$  and generate substi-
  tution boxes  $\sigma_1 \dots \sigma_p$ 
apply substitution boxes  $\sigma_1 \dots \sigma_p$  to input  $l_1 l_2 \dots l_n$  and
  compute  $c_1 c_2 \dots c_n$ 
call procedure 4 with input  $n$  to build random permutation
   $\tau$  on  $\{1, 2, \dots, n\}$ 
permute  $c_1 \dots c_n$  to  $c_{\tau(1)} \dots c_{\tau(n)}$ 
call procedure 5  $p$  times on input  $n$  and generate substi-
  tution boxes  $\alpha_1 \dots \alpha_p$ 
apply  $\alpha_1 \dots \alpha_p$  to input  $c_{\tau(1)} \dots c_{\tau(n)}$ 
Output:  $o_1 o_2 \dots o_n$ 

```

In machine procedure 6, FIG. 11 shows an embodiment of hiding bits with a permutation (secret map) performed in the loop

```

while  $j \leq k$ 
{
  set bit location  $l_{p(j)}:=b_j$ 
  increment  $j$  by 1
}

```

In an embodiment, permutation p is implemented with a Butterfly network in hardware. (A Butterfly network is shown in FIG. 18.)

13.2 Hiding Instruction Order

Machine procedure 7 hides the hidden stable instructions, constructed by machine procedure 6, inside a block of size b containing dummy instructions. b is the sum of the number of stable instructions and the number of dummy instructions in the block. m is the maximum number of stable instructions hidden. γ is a random permutation on $\{1, 2, \dots, b\}$. S_1, S_2, \dots, S_m are a sequence of stable instructions that are part of the program.

Machine Procedure 7. Hide Stable Instructions in Block

```

Input:  $b, m, \gamma$ , and  $S_1, S_2, \dots$  and  $S_m$ 
use  $q$  to randomly choose a positive
  integer  $c$  in the set  $\{1, 2, \dots, \lfloor b/2 \rfloor\}$ 
set  $i:=1$  set  $j:=1$  set  $k:=1$ 
while  $i \leq b$ 
{
  use  $q$  to randomly choose a positive
    integer  $r$  in the set  $\{1, 2, \dots, b\}$ 
  if  $((r \leq c) \text{ and } (j < m))$ 
  {
    hide instruction  $S_j$  at location  $\gamma(i)$ 
  }
}

```

59

```

    increment j by 1
  }
  else
  {
    randomly build dummy instruction  $D_k$ 
    hide instruction  $D_k$  at location  $y(i)$ 
    increment k by 1
  }
  increment i by 1
}

```

FIG. 17 shows a representation of the stable hidden instructions permuted in the block with dummy instructions after machine procedure 7 is completed. Because c is randomly selected in $\{1, \dots, \lfloor b/2 \rfloor\}$, and m is determined during compilation and m can be randomly selected for each block, Eve does not know how many valid instructions are in a block. Eve does not know the probability distribution of valid versus dummy instructions in a block; and Eve does not know the block size. Eve does not know y , and she does not know how to distinguish a dummy opcode from a valid opcode, since the dummy opcodes are also selected using machine procedure 1.

In our cryptographic model, Alice's Protected Machine Hardware (FIG. 15) should execute the operations in machine procedures 6 and 7 because the sboxes $\sigma_1 \dots \sigma_p$ and $\alpha_1 \dots \alpha_p$ and permutations ρ, τ, γ "act as Alice's private keys."

13.3 Executing Stable, Hidden Instructions

After reading one or more hidden machine instructions from memory system 198 in FIG. 1F (Unprotected Memory 1520 in FIG. 15), there are three steps to executing a stable, hidden instruction in Protected Memory Hardware 1502 in FIG. 15.

1. Unmask and find the nearest valid opcode.
2. Extract the operands from the noise.
3. Execute a valid instruction.

In our cryptographic model, our hardware embodiments execute all three steps in the blue region of Protected Machine Hardware, shown in FIG. 15. Our model assumes Eve does not have access to the internal physical operations in the blue region during the execution of these three steps.

With our model assumption about Eve, we proceed to examine steps 1, 2, and 3 in more detail. We developed a machine (compilation tool), called SM (stable machine), in ANSI C that performs these 3 steps. In general, our SM tool can execute any hidden program that operates according to the register machine instructions, described in section 12.1. After a brief summary of steps 1, 2, and 3, we demonstrate an example of our SM tool, executing our greatest common divisor program with hidden, stable register machine instructions.

We assume that our cryptographically stable instructions are stored in Unprotected Memory, as shown in FIG. 15. After the hidden instructions are retrieved by the Protected Machine Hardware from Unprotected Memory (FIG. 15), they are unmasked and executed in the Protected Machine Hardware. In step 1, the first argument of the instruction is a noisy opcode. Our SM tool finds the nearest valid opcode to the noisy opcode, by computing the Hamming distance between the noisy opcode and valid opcodes. In step 2, the operands are extracted from the noise by executing a procedure that performs the inverse of machine procedure 6. In

60

step 3, a valid opcode executes with unmasked operands as input in the Protected Machine Hardware (FIG. 15).

Example 2. Initial GCD Program

```

5  SET A 6
   SET B 10
   IF A 0; Instruction 2
   JMP 12
10 IF B 0
   JMP 13
   IFGT A B
   SUB A B
   IFGT B A
15 SUB B A
   IFN A B
   JMP 2
   SET A B; Instruction 12
   SET B A; Instruction 13
20 GCD Instructions Executed:
   SET A 6
   SET B 10
   IF A 0
   IF B 0
25 IFGT A B
   IFGT B A
   SUB B
   IFN A B
   JMP 2
30 IF A 0
   IF B 0
   IFGT A B
   SUB A B
   IFGT B A
35 SUB B A
   IFN A B
   SET A B
   SET B A

```

After the last instruction SET B A executes, both registers A and B are storing 2. Observe that the instruction numbers in an SM program always start at 0, so JMP 2 causes the machine to execute IF A 0.

Next, we show what our GCD program looks like to Eve in Unprotected Memory after two distinct compilations, generated by our SM tool. In compilations 1 and 2, we selected input $n=64$ for machine procedure 1, and inputs $l=25$, $m=12$, and $n=64$ for machine procedure 3. In compilation 1 and 2, we selected input $n=128$ for machine procedure 4; input $n=8$ for machine procedure 5; and inputs $k=64$ and $n=128$ for machine procedure 6. This means the first and the second operands are hidden in 128 bits: 64 bits of signal and 64 bits of noise.

In compilation 1, the unmasked instruction IF A 0 is represented by the stable, hidden instruction:

```

55 0x88dc21590a7a2106      0xb702d0d37001774f
    0xc92f38e3d0055f3b      0x474bae9b755073f2
    0xd4c61a40b2933bd
Compilation 1:
    0xe26e1277d3a579bf      0x39f2c953e7046f23
60 0x98be08f00998941      0x43fdaf6de036e294
    0x67aea194b64513fe      0x99a72e62201e3930
    0x17739ec80b9eca31      0x9909893bf8361315
    0x13ba2ba4b91a8c32      0xf22ebbf9b308fc15
    0x88dc21590a7a2106      0xb702d0d37001774f
    0xc92f38e3d0055f3b      0x474bae9b755073f2
65 0xd4c61a40b2933bd      0x5e18ef161ee7b6a1
    0x57d0443843e44fb7      0x19511ec8dbdcb1d8

```

61

0xa96c37607d2d54a3
 0x328f4d59c5bc31ba
 0x93670351407ff7da
 0x28ca0548b0892232
 0x63152d2897cbfaeb
 0x2be42c77940ff112
 0x11f70d8b87bda89e
 0xcac54aacb33c81fe
 0x249c758581e55115
 0xa776d331880d488d
 0xcd282b7a697566e4
 0xa19a68022297dcc9
 0x1962c31f19c647da
 0xa631af048aa87dd2
 0x594589c2080ba731
 0xa97b76658999020
 0x4e323afe9b11f7a3
 0x1ab800875b1dcf61
 0xea28b11c25e19c1b
 0x879486810db85603
 0x322f3ae8b4875f31
 0xdd0bd362231e62b0
 0xc822f0cc1804314a
 0x68de174ab9456e7b
 0x49578a0ad2eabf67
 0xbflbdf7bc36d313 0xb1c5ab7d12e09813

Compilation 2:
 0xae9650c9511ba48
 0x9f16b4db47ba5e2
 0x6127f139c6ab535f
 0xfe0c045818dacaaf
 0x5c05485d970005fa
 0x4bcb256be98d9202
 0x3312667ff99273d2
 0x6e9701310cd75a60
 0xe682cee565c93be
 0x5277aadccab828be
 0x967b02174ed59218
 0x6e2ea2ef283666ca
 0x807e3da180ff51a4
 0xe95a7e1bb96c1a90
 0x766fe6187bb03882
 0x3f4134abaa27c677
 0xf44bf817201b738a
 0x97a1be2a1b89d925
 0xb9cf5ede087e7b22
 0x5cb5f49dd0f918ae
 0x28e251135cb6c663
 0x5cd447bddbf7988
 0xe57ccc3231efb080
 0xd175452d458a28a0
 0xfccc9c68a5a6b997
 0xe0db8bb402bf9b1f
 0x92b52b7d38f2ef42
 0xe451d3a900d115c1
 0x2f116f0f6d83aa98
 0xd564e62dad318893
 0x35ccbd0c2467069b
 0x19bcd19c97ae230
 0xe0e906f3b593596b
 0x9928555a8ffbc18f
 0x4d66041d7e35a83b 0x9a16a8204dddbdc4c

In compilation 2, the unmasked instruction IF A 0 is represented by the stable, hidden instruction:
 0x4bcb256be98d9202
 0x3312667ff99273d2
 0x6e9701310cd75a60

62

The first part 0x4bcb256be98d9202 (64 bits) hides the opcode IF; the second and third blocks 0xcde07c9d5cc3f829 0x3312667ff99273d2 (64 bits each) hide the first operand A; the fourth and fifth blocks 0x714e581cb4683b7a 0x6e9701310cd75a60 hide the second operand 0.

As an example of the effectiveness of our hiding procedures, the Hamming distance between the 64-bit hidden opcode 0xe26e1277d3a579bf in compilation 1, representing SET in unmasked instruction SET A 6, and the 64-bit hidden opcode 0xae9650c9511ba48 in compilation 2, also representing SET, is 39. The expected Hamming distance for two hidden opcodes that are statistically independent is 32. The standard deviation is $2/\sqrt{64}$, so 39 is less than two standard deviations away from the expected Hamming distance.

The hiding is also effective for two of the same unmasked opcodes in the same compilation: in compilation 1, the Hamming distance $d(0xe26e1277d3a579bf, 0x99a72e62201e3930)=35$, where 0xe26e1277d3a579bf represents SET in SET A 6, and 0x99a72e62201e3930 represents SET in SET B 10.

14 STATISTICAL TESTING FOR TWO ATTACKS

The first test searches for a birthday paradox attack with distinct SM compilations of a fixed 64-bit random opcode in the unmasked instruction SET A 6.

We used $n=64$ for procedure 1 to generate the random opcode for SET, and inputs $l=12$, $m=12$, and $n=64$ for procedure 3. We used input $n=128$ for procedure 4; and input $n=8$ for procedure 5; and inputs $k=64$ and $n=128$ for procedure 6. We searched for collisions, where 75% of the bits match: that is, where the Hamming distance was greater than 96 for two opcode portions of the instruction SET A 6.

The statistics for 44850 Birthday paradox collision search comparisons are shown below, where each tally is the hamming distance between two 128-bit hidden opcodes, generated from instruction SET A 6. Let h_i be the number of pairs of distinct stable instructions whose 128 bits of opcode are a Hamming distance of i bits apart. Between the two dashed lines is the data for a typical run with 300 stable instructions generated from SET A 6.

TOTAL number of Hamming distances h_i computed=44850.

$h_{40}=1$. $h_{41}=1$. $h_{42}=0$. $h_{43}=2$. $h_{44}=4$. $h_{45}=6$. $h_{46}=14$. $h_{47}=33$.
 $h_{48}=64$. $h_{49}=87$. $h_{50}=150$. $h_{51}=237$. $h_{52}=328$. $h_{53}=460$.
 $h_{54}=667$. $h_{55}=902$. $h_{56}=1114$. $h_{57}=1451$. $h_{58}=1809$.
 $h_{59}=2142$. $h_{60}=2516$. $h_{61}=2795$. $h_{62}=2927$. $h_{63}=3068$.
 $h_{64}=3157$. $h_{65}=3184$. $h_{66}=2929$. $h_{67}=2729$. $h_{68}=2492$.
 $h_{69}=2100$. $h_{70}=1809$. $h_{71}=1522$. $h_{72}=1115$. $h_{73}=929$.
 $h_{74}=676$. $h_{75}=480$. $h_{76}=361$. $h_{77}=219$. $h_{78}=145$.
 $h_{79}=93$. $h_{80}=57$. $h_{81}=38$. $h_{82}=18$. $h_{83}=12$. $h_{84}=1$. $h_{85}=6$.
 When $i>85$, $h_i=0$.

Empirical mean=64.02. $\mu=64$. Expected standard deviation $\sigma=5.66$.

30039 h_i are within σ of μ . Expected h_i within $\sigma=30615$.
 42973 h_i are within 2σ of μ . Expected h_i within $2\sigma=42805$.
 44714 h_i are within 3σ of μ . Expected h_i within $3\sigma=44733$.

No Hamming distances were close to 96 (75% of 128).

In test 2, we searched for an attack by looking at multiple compilations of the same instruction at different locations in a single SM compilation of the program. To simplify this search and make an attack easier to find, we built a program with 150 identical instructions: SET A 6. After one compi-

lation, we computed pairwise Hamming distances. Statistics for a typical run of test 2 are shown below between the two dashed lines.

TOTAL number of Hamming distances h_i computed=11175.

$h_{43}=4$. $h_4=2$. $h_{45}=3$. $h_{46}=4$. $h_{47}=7$. $h_{48}=15$. $h_{49}=30$.
 $h_{50}=43$. $h_{51}=43$. $h_{52}=75$. $h_{53}=122$. $h_{54}=167$. $h_{55}=237$.
 $h_{56}=309$. $h_{57}=389$. $h_{58}=467$. $h_{59}=520$. $h_{60}=648$.
 $h_{61}=67$. $h_{62}=701$. $h_{63}=753$. $h_{64}=867$. $h_{65}=782$.
 $h_{66}=697$. $h_{67}=670$. $h_{68}=655$. $h_{69}=535$. $h_{70}=437$.
 $h_{71}=345$. $h_{72}=259$. $h_{73}=223$. $h_{74}=168$. $h_{75}=117$.
 $h_{76}=82$. $h_{77}=46$. $h_{78}=37$. $h_{79}=23$. $h_{80}=10$. $h_{81}=7$. $h_{82}=6$.
 $h_{83}=1$. $h_{84}=1$. $h_{85}=1$. When $i>85$, $h_i=0$.

Empirical mean=63.90. $\mu=64$. Expected standard deviation $\sigma=5.66$

7495 h_i are within σ of μ . Expected h_i within $\sigma=7628$.

10735 h_i are within 20 of μ . Expected h_i within 20
 $\sigma=10665$. 11139 h_i are within 3σ of μ . Expected h_i
within $3\sigma=11146$.

No Hamming distances were close to 96; the statistics follow a binominal distribution with

$$p = \frac{1}{2}.$$

In general, we do not expect this attack to be effective for Eve because Alice should not voluntarily compile her plaintext code with static “keys” on plaintext code that repeats the same instruction multiple times on purpose. Furthermore, Eve’s potential collisions occur only on “keys” that Eve artificially constructs with the SM tool. In a proper use setting, Alice uses a different set of “keys” and randomized opcodes on each separate compilation on a particular machine, and procedure 7 further reduces the efficacy of this type of attack.

Although the invention(s) have been described with reference to specific embodiments, it will be understood by those skilled in the art that various changes may be made and equivalents may be substituted for elements thereof without departing from the true spirit and scope of the invention. In addition, modifications may be made without departing from the essential teachings of the invention.

REFERENCES

- [1] Wikipedia. Transmission Control Protocol/Internet Protocol.
- [2] Claude Shannon. Communication Theory of Secrecy Systems. 1949.
- [3] Charles H. Bennett and Gilles Brassard. Quantum cryptography: Public key distribution and coin tossing. In Proceedings of IEEE International Conference on Computers, Systems and Signal Processing. 175, 175-179. New York, 1984.
- [4] Charles H. Bennett, François Bessette, Gilles Brassard, Louis Salvail and John Smolin. Experimental Quantum Cryptography. Journal of Cryptology. 5, No. 1, 3-28, 1992.
- [5] P. D. Townsend, J. G. Rarity and P. R. Tapster. Single photon interference in a 10 km long optical fibre interferometer. Electronic Letters. 29, No. 7, 634-635, April 1993.
- [6] P. D. Townsend, J. G. Rarity and P. R. Tapster. Enhanced single photon fringe visibility in a 10 km long prototype quantum cryptography channel. Electronic Letters. 29, No. 14, 1291-1293 July 1993.

- [7] Nwokedi Idika and Aditya P. Mathur. “A Survey of Malware Detection Techniques.” Technical Report. Dept. of Computer Science. Purdue Univ., February 2007.
- [8] Andreas Moser, Chris Kruegel and Engin Kirda. “Limits of Static Analysis for Malware Detection.” IEEE. 23rd Annual Computer Security Applications Conf., 2007.
- [9] Diego Zamboni (editor). Proc. of the 5th Intl. Conf. on Detection of Intrusions and Malware LNCS. Springer. July 2008.
- [10] John Mitchell and Elizabeth Stillson. Detection of Malicious Programs. U.S. Pat. No. 7,870,610, 2011.
- [11] Fred Cohen. “Computer Viruses Theory and Experiments. Computers and Security.” 6(1) 22-35, February 1987.
- [12] Stephen Cook. The P versus NP Problem. Clay Math Institute, 2013.
- [13] Eric Filiol. “Malicious Cryptology and Mathematics.” Cryptography and Security in Computing. Ch. 2. Intech. Mar. 7, 2012.
- [14] Eric Filiol. Computer Viruses: From Theory to Applications. Springer, 2005.
- [15] Cheryl Pellerin. DARPA Goal for Cybersecurity: Change the Game. American Forces Press, Dec. 20, 2010.
- [16] Harold Abelson and Gerald J. Sussman with J. Sussman. Structure and Interpretation of Computer Programs. 2nd Edition, MIT Press, 1996.
- [17] John Hennessy and David Patterson. Computer Architecture. 5th Edition, Morgan Kaufmann, 2012.
- [18] Aleksandr Andronov and Lev Pontrjagin. “Systèmes Grossiers.” Dokl. Akad. Nauk., SSSR, 14, 247-251, 1937.
- [19] D. Anosov. “Geodesic flows on closed Riemannian manifolds of negative curvature.” Proc. Steklov. Inst. Math. 90, 1967.
- [20] Jacob Palis and Stephen Smale. “Structural Stability Theorems.” Proc. Symp. Pure Math. AMS. 14, 223-232, 1970.
- [21] Keonhee Lee and Kazuhiro Sakai. “Structural stability of vector fields with shadowing.” Journal of Differential Equations. 232, 303-313, 2007.
- [22] Clark Robinson. Dynamical Systems. Stability, Symbolic Dynamics and Chaos. CRC Press, 1996.
- [23] James R. Munkres. Topology. Prentice-Hall, 1975.
- [24] Leigh Van Valen. An Evolutionary Law. 1-30, 1973.
- [25] Craig Gentry. A Fully Homomorphic Encryption Scheme. Ph.D. Thesis, Stanford University, 2009.
- [26] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachene. TFHE: Fast Fully Homomorphic Encryption over the Torus.
- [27] Ravital Solomon. An Intro to Fully Homomorphic Encryption for Engineers. Apr. 25, 2021. (<https://ravital.github.io>)
- [28] Gaurav S. Kc, et. al. “Countering Code-Injection Attacks With Instruction-Set Randomization.” CCS 2003. ACM, October 27-30, 2003.
- [29] Andrew Yao. “Protocols for Secure Computations.” 23rd Annual Symposium on Foundations of Computer Science. IEEE Computer Society, 160-164, 1982.
- [30] David Evans, et. al. A Pragmatic Introduction to Secure Multi-Party Computation. NOW, 2021.
- [31] Raul Rojas. “Conditional Branching is not Necessary for Universal Computation in von Neumann Computers.” Journal of Universal Computer Science. 2, No. 11, 756-768, 1996.

- [32] Karl S. Lashley. Studies of cerebral function in learning XII. Loss of the maze habit after occipital lesions in blind rats. *Journal of Comparative Neurology*. 79, Issue 3, 431–462, December 1943.
- [33] Karl S. Lashley. In search of the engram. *Society of Experimental Biology Symposium 4: Physiological Mechanisms in Animal Behavior*, 454–482, Cambridge University Press, 1950.
- [34] Brian Kernighan and Dennis Ritchie. *The C Programming Language*. 2nd Edition, Prentice Hall, 1988.
- [35] Gilbert Vernam. Cipher printing telegraph systems for secret wire and radio telegraphic communications. *J. Americ. Inst. Elect. Eng.* 55, 109–15, 1926.
- [36] Alan M. Turing. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. Series 2* 42 (Parts 3 and 4), 230–265, 1936. A correction, *ibid.* 43, 544–546, 1937.
- [37] Simon Kochen and Ernst P. Specker. The Problem of Hidden Variables in Quantum Mechanics. *Journal of Mathematics and Mechanics (now Indiana Univ. Math Journal)* 17 No. 1, 59–87, 1967.
- [38] John Conway and Simon Kochen. The Strong Free Will Theorem. *Notices of the American Mathematical Society*. 56(2), 226–232, February 2009.
- [39] NIST. Advanced Encryption Standard (AES), FIPS 197. November 2001.
- [40] Ross Anderson, Eli Biham, Lars Knudsen. A Proposal for the Advanced Encryption Standard.
- [41] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*. 21, 120–126, 1978.
- [42] Klint Finley. Chinese Supercomputer Is Still the World's Most Powerful. *Wired Magazine*. Nov. 18, 2013.
- [43] A. F. Webster and S. E. Tavares. On the Design of S-Boxes. *Advances in Cryptology. CRYPTO 85 Proceedings*. LNCS 218. Springer, 523–534, 1986.
- [44] NIST. FIPS-180-2: Secure Hash Standard, August 2002.
- [45] Guido Bertoni, Joan Daemen, Michael Peeters, Gilles Van Assche. *Keccak Reference 3.0* 2011.
- [46] Jean-Philippe Aumasson, Samuel Neves, Zooko Wilcox-O'Hearn, Christian Winnerlein. *BLAKE*.
- [47] Praveen Gauravaram, Lars Knudsen, Krystian Matusiewicz, Florian Mendel, Christian Rechberger, Martin Schlaffer, and Soren S. Thomsen. *GrØstl—a SHA-3 candidate*.
- [48] Hongjun Wu. *The Hash Function JH*. 2011.
- [49] Niels Ferguson, Stefan Lucks, Bruce Schneier, Doug Whiting, Mihir Bellare, Tadayoshi Kohno, Jon Callas, Jesse Walker. *The Skein Hash Function Family*. 2010.
- [50] Thomas Bayes. An essay towards solving a problem in the doctrine of chances. *Philosophical Transactions of the Royal Society of London*. 53, 370–418, 1764.
- [51] William Feller. *An Introduction to Probability Theory and Its Applications*. Volume II. John Wiley. 1966.
- [52] Harald Cramer. *Mathematical Methods of Statistics*. Princeton University Press. 1946.
- [53] Abraham De Moivre. *The Doctrine of Chances: or, A Method of Calculating the Probabilities of Events in play*. 1st Edition London, 1718; 2nd edition 1738; 3rd edition 1756.
- [54] Ralph C. Merkle. Secure Communications over Insecure Channels. *Communications of the ACM*. 21 (4), 294–299, April 1978.
- [55] Whitfield Diffie and Martin Hellman. New directions in cryptography. *IEEE Transactions on Information Theory* 22, 644–654, 1976.

- [56] Joseph H. Silverman and John Tate. *Rational Points on Elliptic Curves*. Springer-Verlag, 1992.
- [57] Harold Edwards. A normal form for elliptic curves. *Bulletin of the American Mathematical Society*. 44: 393–422, April 2007.
- [58] Peter Montgomery. Speeding the Pollard and Elliptic Curve Methods of Factorization. *Mathematics of Computation* 48 (177): 243–264, 1987.
- [59] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, N Y, 1997.
- [60] P. C. van Oorschot and M. J. Wiener. On Diffie-Hellman Key Agreement with Short Exponents. *Eurocrypt 1996*, LNCS 1070, Springer, 332–343, 1996.
- [61] A. C. Geary. Analysis of a Man-In-The-Middle-Attack on the Diffie-Hellman Key Exchange Protocol. NPS, 2009.
- [62] Peter W. Shor. Algorithms for quantum computation: discrete log and factoring. In *Proceedings of the 35th Annual IEEE Symposium on Foundations of Computer Science*, pages 2–22, 1994.
- [63] John Proos and Christof Zalka. Shor's discrete logarithm quantum algorithm for elliptic curves, 2003.
- [64] Mihir Bellare, Ran Canetti and Hugo Krawczyk. Keying Hash Functions for Message Authentication. *Advances in Cryptology-Crypto 96 Proceedings*. LNCS 1109, N. Koblitz ed., Springer, 1996.
- [65] Mark Wegman and J. Lawrence Carter. New Hash Functions and Their Use in Authentication and Set Equality. *Journal of Computer and System Sciences*. 22, 265–279, 1981.
- [66] Serge Vaudenay. Secure Communications over Insecure Channels Based on Short Authenticated Strings. *Advances in Cryptology CRYPTO 2005*. 309–326. 2005.
- [67] Richard W. Hamming. "Error Detecting and Error Correcting Codes." *The Bell System Technical Journal*. 29(2). 117–160. April 1950.
- [68] Zhijie Jerry Shi and Ruby B. Lee. Implementation Complexity of Bit Permutation Instructions. *Proceedings of the Asimolar Conference on Signals, Systems, and Computers*. 879–886. 2003.

The invention claimed is:

1. A machine-implemented method of randomizing machine instructions comprising:
 - a machine having a processor system and a memory system, the processor system including one or more processors;
 - constructing random opcodes wherein each opcode is represented with one or more bits;
 - the machine computing a Hamming distance between the random opcodes;
 - the machine flipping the bits of one or more of the opcodes until each pair of distinct opcodes are at least a Hamming distance of 2l+1 apart wherein l is a non-negative natural number;
 - a machine generating noise;
 - wherein the bits of each random opcode are hidden inside of the noise.
2. The machine-implemented method of claim 1 further comprising:
 - at least one random opcode represents a conditional branch instruction.

67

3. The machine-implemented method of claim 1 further comprising:

at least one random opcode represents an unconditional branch instruction.

4. The machine-implemented method of claim 1 further comprising:

at least one random opcode represents an instruction that adds two numbers.

5. The machine-implemented method of claim 1 further comprising:

at least one random opcode represents an instruction that subtracts two numbers.

6. The machine-implemented method of claim 1 further comprising: the noise is generated by measuring quantum events due to photons.

7. The machine-implemented method of claim 6 further comprising: the photons are emitted from a light emitting diode.

8. The machine-implemented method of claim 7 further comprising:

the measurement of quantum events is based upon the arrival times of emitted photons.

9. The machine-implemented method of claim 1, wherein the constructing of the random opcodes uses a non-deterministic method.

10. The machine-implemented method of claim 1, wherein the constructing of the random opcodes uses a deterministic method.

11. The machine-implemented method of claim 1, wherein the bit locations of the random opcode hidden in the noise are determined from one or more permutations.

12. The machine-implemented method of claim 1, wherein one or more operands corresponding to the opcode are hidden in noise, wherein the bit locations of the one or more operands are determined from one or more permutations.

68

13. A machine-implemented method of executing machine instructions comprising:

a machine having a processor system and a memory system, the processor system including one or more processors;

the machine extracts a machine instruction that contains an opcode and one or more operands, wherein the opcode and one or more operands are hidden in noise; wherein each opcode is represented with one or more bits; the machine computing a Hamming distance between the extracted opcode and a set of valid opcodes;

wherein the machine selects a valid opcode from the set that is closest to the extracted opcode;

wherein the machine executes the instruction represented by the valid opcode and one or more operands.

14. The method of claim 13 further comprising:

at least one valid opcode represents a conditional branch instruction.

15. The method of claim 13 further comprising:

at least one valid opcode represents an unconditional branch instruction.

16. The method of claim 13 further comprising:

at least one valid opcode represents an instruction that adds two numbers.

17. The machine-implemented method of claim 13, wherein one or more permutations are used to extract the opcode that is hidden in noise.

18. The machine-implemented method of claim 13, wherein one or more permutations are used to extract the one or more operands that are hidden in noise.

19. The machine-implemented method of claim 13, wherein the valid opcode represents a machine instruction that multiplies two numbers.

20. The machine-implemented method of claim 13, wherein the valid opcode represents a machine instruction that copies memory from one address of the machine to another address of the machine.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 12,124,850 B2
APPLICATION NO. : 17/516722
DATED : October 22, 2024
INVENTOR(S) : Michael Stephen Fiske

Page 1 of 2

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

In the Specification

Column 54 Line 36: Replace $21+1$ with $2L+1$

Column 54 Line 38: Replace 1 with L

Column 54 Line 40: Replace radius 1 with radius L

Column 54 Line 40: Replace 1) with L)

Column 55 Line 7: Replace $==$ with $:=$

Column 55 Line 10: Replace $==$ with $:=$

Column 55 Line 14: Replace $==$ with $:=$

Column 55 Line 24: Replace $21+1$ with $2L+1$

Column 55 Line 26: Replace 1 with L

Column 55 Line 31: Replace $21+1$ with $2L+1$

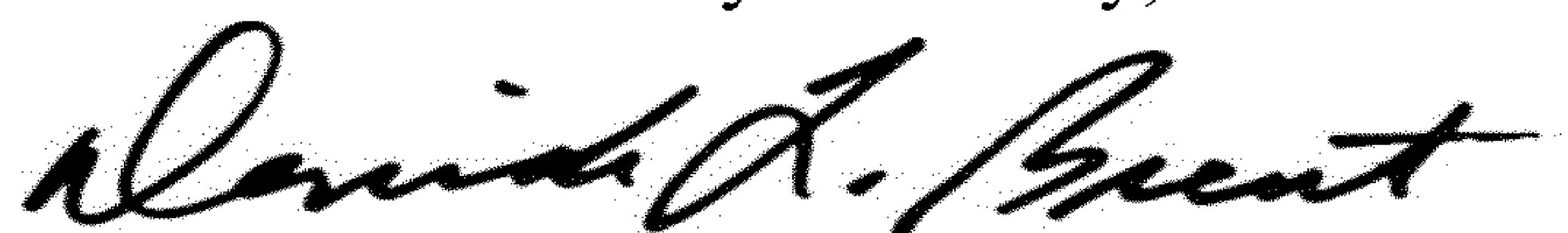
Column 55 Line 33: Replace $21+1$ with $2L+1$

Column 55 Line 43: Replace $21+1$ with $2L+1$

Column 55 Line 47: Replace $21+1$ with $2L+1$

Column 55 Line 48: Replace $21+1$ with $2L+1$

Signed and Sealed this
Fourteenth Day of January, 2025



Derrick Brent
Acting Director of the United States Patent and Trademark Office

CERTIFICATE OF CORRECTION (continued)
U.S. Pat. No. 12,124,850 B2

Page 2 of 2

Column 55 Line 49: Replace $21+1$ with $2L+1$

Column 55 Line 53: Replace If 1 with If L

Column 55 Line 53: Replace $(21+1)$ with $(2L+1)$

In the Claims

Column 66 Line 58 Claim 1: Replace $21+1$ apart wherein 1 with $2L+1$ apart wherein L